A FAST AVERAGE CASE ALGORITHM FOR LYNDON DECOMPOSITION

Costas S. Iliopoulos

Department of Computing
King's College, University of London
London WC2R 2LS England

School of Computing
Curtin University of Technology
Perth WA 6001 Australia

W. F. Smyth

Department of Computer Science & Systems
McMaster University
Hamilton, Ontario L8S 4K1 Canada

School of Computing
Curtin University of Technology
Perth WA 6001 Australia

ABSTRACT

A simple algorithm, called LD, is described for computing the Lyndon decomposition of a word of length n. Although LD requires time $O(n\log n)$ in the worst case, it is shown to require only $\Theta(n)$ worst-case time for words which are "1-decomposable", and $\Theta(n)$ average-case time for words whose length is small with respect to alphabet size. The main interest in LD resides in its application to the problem of computing the canonical form of a circular word. For this problem, LD is shown to execute significantly faster than other known algorithms on important classes of words. Further, experiment suggests that, when applied to arbitrary words, LD on average outperforms the other known canonization algorithms in terms of two measures: number of tests on letters and execution time.

KEYWORDS

combinatorial, algorithm, word, string, monoid, Lyndon, decomposition, factorization, canonization, canonical form, lexicographically least, circular, average case.

AMS SUBJECT CLASSIFICATION

68C25.

1 INTRODUCTION

Let A be a non-empty set of distinct totally ordered elements. We call A an *alphabet* of *size* $\alpha = |A|$, and we call its elements *letters*. Denote by A^+ the set of all *words* which can be formed from letters of A by repeated *concatenation*. Then, for example, if $A = \{a\}$, $A^+ = \{a^i \mid i = 1, 2, \ldots\}$, where a^i denotes the concatenation of i a's. Let $A^* = A^+ \cup \{\epsilon\}$, where ϵ is the empty word. A^+ and A^* are ordered *lexicographically*.

Consider a word $w \in A^+$ which contains n concatenated letters. We say that w has *length* $n = |w|$. If for some $u \in A^+$, $v \in A^*$, it is true that $w = uv$, then u is called a *prefix* of w. If moreover $v \neq \epsilon$, then u is called a *proper prefix* of w. (A *suffix* and a *proper suffix* are defined analogously.) w is called a *Lyndon word* iff for every proper prefix u such that $w = uv$, it is true that $w < vu$. Denote by L the set of Lyndon words. Clearly $L \subseteq A^+$ and, for example, if $A = \{a, b\}$, then $\{a, b, ab, aab\} \subset L$ and $\{aa, ba, aba, abab\} \subset A^+ \backslash L$.

Lyndon words are of interest primarily because of the following remarkable theorem [CFL58]:

Theorem 1.1 Every word $w \in A^+$ has a unique decomposition $w = w_1w_2 \dots w_m$ into Lyndon words w_i, $i = 1, 2, \dots, m$, satisfying $w_1 \geq w_2 \geq \cdots \geq w_m$. $\diamondsuit$

It turns out that computing the Lyndon decomposition of a given word w of length n is closely related to other problems [D83]:

* computation of the lexicographically least rotation (canonical form) of a "circular" word formed from w;
* computation of a minimum/maximum suffix of w.

The first of these problems has rather unexpected applications in computational geometry (determining whether or not given polygons are similar [AT78,IS89]) and graph theory (determining whether or not given cycles have the same degree sequence [CB81]). The next two problems mentioned are well-known and arise in various contexts [AHU74, 329ff]. All of these problems are solved by Duval's Lyndon decomposition algorithm [D83], which executes in time $\Theta(n)$.

In Section 2 of this paper we present a new Lyndon decomposition algorithm called LD and show that it executes in time $O(n\log n)$ using $O(\log n)$ space. In Section 3 we show that, on certain classes of words, LD executes in average time $\Theta(n)$ and, furthermore, requires on average less than $n + \log_2 n$ tests on the letters in the word. In Section 4 we introduce the circular word canonization problem and show, once again over certain classes of words, that a version of LD called FMSP solves this problem using fewer letter tests than other available algorithms. We also give the results of experimental comparisons between FMSP and three other algorithms, including Duval's, which solve it: our experiments indicate that on average FMSP is superior to the other algorithms, not only in terms of letter tests, but also in terms of execution time.

2 ALGORITHM LD

The strategy of Algorithm LD is based on the observation that computing the Lyndon decomposition of w is easy if, for each prefix u of w, there is exactly one position in u which contains the minimum letter of u. It is not difficult to see that if a word w has this property, then every Lyndon word in its Lyndon decomposition has a distinct first letter. We say therefore that such words w are *1-decomposable*. Examples of 1-decomposable words are the Lyndon word $w = abc \dots xyz$, where the minimum letter of each prefix is always in position 1; $w = zyx \dots cba$, where the minimum letter of each prefix is always its rightmost letter; and $w = cddbccabbcc$, which decomposes into $w_1 = cdd$, $w_2 = bcc$, and $w_3 = abbcc$. The strategy of LD is simply to deal very efficiently with 1-decomposable words and to handle the more difficult ones as special cases. We now introduce notation and terminology which allow this strategy to be described precisely.

Since it is often necessary to refer to subwords of a given word w, it becomes convenient to represent w as an array W. Then for given integers i and j satisfying $1 \leq i \leq j \leq n$, we let $W[i,j]$ denote the subword of W beginning at position i and ending at position j. When $j = i$, $W[i,i]$ is usually abbreviated to $W[i]$.

Given W and an integer $j \in [1,n]$, we say that a positive integer $i \leq j$ is *j-feasible* iff for every integer $h = 0, 1, \dots, j-i$, $W[i, i+h]$ is a minimum over all subwords of length $h+1$ of $W[1,j]$. As we shall see below (Lemma 2.2), if i is not j-feasible for any j, then no Lyndon word in the Lyndon decomposition of W begins at position i. For any $j \in [1,n]$ let

$$F_j = \{i_1, i_2, \dots, i_k\} \qquad \dots (2.1)$$

denote the set of all j-feasible positions of $W[1,j]$, arranged in ascending order so that always $i_1 < i_2 < \cdots < i_k$. Then in particular $F_1 = \{1\}$. Algorithm LD executes by proceeding from left to right through W, thus successively letting j take the values $1, 2, \dots, n$, and computing F_j at each step. In order to understand this process, we require certain elementary properties of F_j:

Property 2.1 $k \geq 1$ (that is, $F_j \neq \emptyset$).

Proof For every integer $j \in [1,n]$, there must exist at least one position i such that $W[i]$ is a minimum over all letters of $W[1,j]$. Then suppose that for some integer $h \geq 1$, there exist $k_h \geq 1$ positions $i_1, i_2, \dots, i_{k_h}$ such that, for every $i \in [1,k_h]$, $W[i, i+h-1]$ is a minimum over all subwords of $W[1,j]$

of length h. It follows that for at least one of these k_h positions, say i', $W[i', i'+h]$ is a minimum over all subwords of $W[1, j]$ of length $h+1$. Then the result follows by induction. $\diamondsuit$

Property 2.2 Suppose F_j is given as in (2.1). Then for all integers r and s satisfying $1 \leq r \leq s \leq k$, and for every nonnegative integer $h \leq j - i_s$,

$$W[i_r + h] = W[i_s + h].$$

Proof Since, by the definition of j-feasible, all the subwords specified by (2.1) are minimum subwords of length h in $W[1, j]$, they must therefore be equal. $\diamondsuit$

Property 2.3 If in (2.1) $j > i_k$, then $F_j \subseteq F_{j-1}$; if $j = i_k$, then $F_j \subseteq F_{j-1} \cup \{j\}$.

Proof Suppose that $j > i_k$. It follows that $W[j] > W[i_k] = W[i_{k-1}] = \cdots = W[i_1]$, and it follows as well that every element of F_j must also be an element of F_{j-1}, as required. Similarly, when $j = i_k$, every element of F_j, excluding j, must also be an element of F_{j-1}. $\diamondsuit$

From these properties it begins to appear that F_{j+1} can be derived from F_j by a kind of refinement process. The following two lemmas make this transition clearer:

Lemma 2.1 Suppose that F_j is defined by (2.1) and that $j < n$. Then

(a) $F_{j+1} = \{j+1\} \iff W[j+1] < W[i_1]$;
(b) $j+1 \in F_{j+1} \iff W[j+1] \leq W[i_1]$;
(c) for every integer $h \in [2, k]$:
  (i) $i_h \in F_{j+1} \iff W[j+1] \leq W[i_1 + (j+1-i_h)]$;
  (ii) if $W[j+1] < W[i_1 + (j+1-i_h)]$, then for every positive integer $h' < h$, $i_{h'} \notin F_{j+1}$.

Proof Since we suppose that F_j has already been determined, it follows that, for every integer $h \in [2, k]$, $W[i_h, j] = W[i_1, i_1 + (j - i_h)]$ is the minimum subword of length $j+1-i_h$. Thus, to determine whether or not $i_h \in F_{j+1}$, it suffices to test $W[j+1]$ against $W[i_1 + (j+1-i_h)]$.

(a) By the definition of j-feasible, every element $W[i_h]$, $1 \leq h \leq k$, is a minimum element of $W[1, j]$. Then if $W[j+1] < W[i_1]$, it follows that no element of F_j can belong to F_{j+1}, hence by Property 2.3 that $F_{j+1} = \{j+1\}$. Suppose on the other hand that $W[j+1] \geq W[i_1]$. Then think of comparing $W[j+1]$ with $W[i_1 + (j+1-i_k)]$ — that is, comparing the letters displaced by the same number of positions $(j+1-i_k)$ from i_k and i_1, respectively. Three cases arise. If $W[j+1]$ is greater, then $i_1 \in F_{j+1}$; if less, then $i_k \in F_{j+1}$; and if the two letters are equal, then $\{i_1, i_k\} \subseteq F_{j+1}$. In each case, $F_{j+1} \neq \{j+1\}$.

(b) Suppose that $j+1 \in F_{j+1}$. But then if $W[j+1] > W[i_1]$, $j+1$ cannot be $(j+1)$-feasible, a contradiction. Hence $W[j+1] \leq W[i_1]$. Conversely, suppose that $j+1 \notin F_{j+1}$. By (a), $W[j+1] \geq W[i_1]$. But if $W[j+1] = W[i_1]$, then $j+1$ is $(j+1)$-feasible, so that by definition $j+1 \in F_{j+1}$, a contradiction. Hence $W[j+1] > W[i_1]$.

(c) (i) If $W[j+1] \leq$ (respectively, $>$) $W[i_1 + (j+1-i_h)]$, then $W[i_h, j+1] \leq$ (respectively, $>$) $W[i_1, i_1 + (j+1-i_h)]$, so that $i_h \in F_{j+1}$ (respectively, $i_h \notin F_{j+1}$).
  (ii) Since $W[j+1] < W[i_1 + (j+1-i_h)]$, $W[i_h, j+1] < W[i_1, i_1 + (j+1-i_h)]$. But for every positive $h' < h$,

$$W[i_1, i_1 + (j+1-i_h)] = W[i_{h'}, i_{h'} + (j+1-i_h)],$$

  from which the result follows. $\diamondsuit$

In order to formulate the next lemma, let us say that a position $i \in [1, n]$ *marks* a Lyndon word of W iff i is the starting position of a Lyndon word of W.

Lemma 2.2 Suppose that F_j is defined by (2.1) and that $j < n$. Suppose further that $W[j+1] < W[i_1]$. Then

(a) $j+1$ marks a Lyndon word of W;
(b) $i \in [i_1, j]$ marks a Lyndon word of W if and only if $i \in F_j$.

Proof To prove (a), observe that $W[j+1]$ is strictly less than any element of $W[1,j]$, and that therefore $j+1$ marks a Lyndon word of length at least one. (And so the previous Lyndon word of W terminates at position j.)

To prove (b), recall that $i \in F_j$ if and only if $W[i,j']$ is a minimum subword of $W[1,j]$ for every $j' \in [i,j]$. For every integer $k' \in [1,k]$, consider $i_{k'} \in F_j$. For $k' < k$, let $j' = i_{k'+1} - 1$; otherwise, for $k' = k$, let $j' = j$. Let j'' denote an integer in the range $[i_{k'}, j]$, and write $W[i_{k'}, j''] = uv$ where u is any proper prefix ($v \neq \epsilon$). Then for $j'' \in [i_{k'}, j']$, $uv < vu$, while for $j'' \in [j'+1, j]$, $uv \geq vu$ as soon as $|u| \geq j' - i_{k'} + 1$. Thus $W[i_1, j]$ decomposes into k non-increasing Lyndon words marked by $i_1, i_2, \ldots, i_k$. Since by Theorem 1.1 the decomposition is unique, (b) follows. $\diamondsuit$

We are now in a position to outline Algorithm LD as a process of computing F_{j+1} from F_j. As already observed, $F_1 = \{1\}$. Then if F_j is given by (2.1), we compute F_{j+1} by comparing $W[j+1]$ with $W[i_1]$. There are three cases:

(1) If less, then by Lemmas 2.1(a) and 2.2(a), $F_{j+1} = \{j+1\}$ and $j+1$ marks a Lyndon word. Further, by Lemma 2.2(b), each element of F_j marks a Lyndon word.

(2) If equal, then by Lemma 2.1(b), $j+1 \in F_{j+1}$ and, by Property 2.3, F_{j+1} is formed by deleting elements from $F_j \cup \{j+1\}$ in accordance with Lemma 2.1(c). (If, during this deletion process, it is found that, for some h, $W[j+1] < W[i_1 + (j+1-i_h)]$, then, in accordance with Lemma 2.2, the elements $i_1, i_2, \ldots, i_{h-1}$ must all mark Lyndon words.)

(3) If greater, then by Property 2.3 F_{j+1} is formed by deleting elements from F_j in accordance with Lemma 2.1(c).

The following algorithm expresses this strategy formally. It outputs, in ascending sequence, the starting positions of the Lyndon words of W.

Algorithm LD1

```
        i1 ← 1; F ← {1}; k ← 1;
        for j ← 2 to n do
            {
            if W[j] ≯ W[i1] then
                if W[j] < W[i1] then
                    {output F; F ← {j}; goto NEXTj}
                else
                    F ← F ∪ {j};
            for h ← k downto 2 do
                if W[j] > W[i1 + (j − ih)] then
                    F ← F − {ih}
                elseif W[j] < W[i1 + (j − ih)] then
                    {output {i1, i2, ..., ih−1};
                    F ← F − {i1, i2, ..., ih−1};
                    goto NEXTj};
NEXTj       k ← |F|
            }
        output F.
```

Note that LD1 executes very efficiently on 1-decomposable words W, since F never contains more than a single element, and so the inner **for** loop is never executed. For such W, the number of tests on letters of W executed by LD1 is exactly $n-1$ plus $m-1$, where m is the number of Lyndon words in W. The trouble with LD1 is that it can also be very slow: it may happen that for $\Theta(n)$ values of j, F_j contains $\Theta(n)$ elements — for instance, when $W = a^n$ or $(ab)^{n/2}$. In such cases, each test in the inner **for** loop is executed $\Theta(n^2)$ times. However, the following result provides a basis for improving the algorithm's worst case complexity:

Lemma 2.3 Suppose that F_j is defined by (2.1). For any integer $r \in [1, k-1]$, let $d_r = i_{r+1} - i_r$. Then for every $r > 1$,

(a) $d_{r-1} \geq d_r$;

(b) $d_{r-1} > d_r \iff d_{r-1} \geq j - i_r + 1$;
(c) $d_{r-1} = d_r \iff W[i_{r-1}, i_r - 1] = W[i_r, i_{r+1} - 1]$;
(d) if $d_{r-1} = d_r$, then $W[i_{r-1}, i_r - 1]$ is a Lyndon word in the Lyndon decomposition of W if and only if $W[i_r, i_{r+1} - 1]$ is also a Lyndon word.

Proof (a) Suppose that $d_{r-1} < d_r$. Then Property 2.2 implies that every ${d_{r-1}}^{\text{th}}$ position after i_r in W must be an element of F_j, hence that there exists an element of F_j between i_r and i_{r+1}, a contradiction.

(b) Sufficiency is trivial. To prove necessity, suppose that $j - i_r + 1 > d_{r-1} > d_r$. Then $i_r + d_{r-1} - 1 < j$, $W[i_r, j] = W[i_{r-1}, j - d_{r-1}]$, and it follows that $i_r + d_{r-1} - 1 = i_s - 1$ for some $s \in [r+2, k]$. We therefore conclude that $W[i_{r-1} + d_r, i_r - 1] = W[i_{r+1}, i_s - 1]$, and so $i_{r-1} + d_r \in F_j$, a contradiction.

(c) Necessity is an immediate consequence of Property 2.2; sufficiency follows from the fact that two subwords can be equal only if they have the same length.

(d) For any integer $s \in [r-1, r]$, let $w_s = W[i_s, i_{s+1} - 1]$. Suppose that w_r is a Lyndon word of W, but that w_{r-1} is not. Then w_{r-1} must be a proper suffix of a Lyndon word $w' = uw_{r-1} \geq w_r$, where $u \neq w_{r-1}$. Since by (c) $w_r = w_{r-1}$, it follows that $u > w_{r-1}$. But then $uw_{r-1} > w_{r-1}u$, contradicting the assumption that w' is a Lyndon word. Hence w_{r-1} must also be a Lyndon word. A similar argument establishes the converse. $\diamondsuit$

Lemma 2.3(a) makes precise the intuitive idea that elements of F_j should normally be less common in the lower-numbered positions of W, since in order to be starting positions of minimum subwords, and thus retained in F_j, these positions have more conditions to satisfy than the higher-numbered positions have. Lemma 2.3(b) strengthens this idea considerably, making clear that if a minimum subword is not immediately duplicated, then it must have a length greater than that of all the subwords determined by the remaining elements of F_j. Further, Lemma 2.3(d) tells us that, over any concatenated sequence of two or more identical subwords, it suffices to record only the starting positions of the leftmost and rightmost subwords in the sequence, together with the length of the subword, since if any of the subwords is a Lyndon word, then they will all be Lyndon words. Let us call each such maximal sequence of identical subwords a *repetition*. Then eliminating unnecessary elements from F_j would mean that any repetition in W would give rise to only two entries in F_j. Further, whenever it occurs that $d_r < d_{r-1}$, then either $r = k - 1$ (marking the last element of F_j) or $d_{r+1} + d_r < d_{r-1}$; in this latter case, it must also be true that $d_{r+1} \leq d_r$, so that $d_{r+1} < d_{r-1}/2$. From this fact it is not hard to argue that, with superfluous elements removed from F_j, $|F_j| \in O(\log n)$. A more detailed analysis enables us to specify an exact upper bound on $|F_j|$. This analysis is given in an Appendix; we state here the result:

Lemma 2.4 Suppose that every repetition in $W[1, j]$ gives rise to at most two entries in F_j. Then

$$k = |F_j| \leq 2 \log_3(n+1).$$

The bound is sharp; more precisely, for every even k, there exists a word of length $n = 3^{k/2} - 1$ for which $|F_n| = k$. $\diamondsuit$

In order to implement the elimination of unnecessary elements from F_j, we redefine F_j as a set of ordered integer pairs (i_h, d_h):

$$F_j = \{(i_1, d_1), (i_2, d_2), \ldots, (i_k, d_k)\}, \qquad \ldots (2.2)$$

where as before the i_h are assumed to be in ascending sequence, and the corresponding d_h are computed from the expression

$$d_h = i_h - i^*,$$

where i^* is the largest j-feasible position less than i_h (in case no such position exists, $d_h = 0$). Then a modified LD algorithm can be described by making a few straightforward changes to LD1:

Algorithm LD2

$i_1 \leftarrow 1;\ d_1 \leftarrow 0;\ F \leftarrow \{(1,0)\};\ k \leftarrow 1;$

```
for j ← 2 to n do
    {
    if W[j] ≯ W[i_1] then
        if W[j] < W[i_1] then
            {output F; F ← {(j, 0)}; goto NEXTj}
        else
            if d_k = j − i_k then i_k ← j
            else F ← F ∪ {(j, j − i_k)};
    for h ← k downto 2 do
        if W[j] > W[i_1 + (j − i_h)] then
            F ← F − {(i_h, d_h)}
        elseif W[j] < W[i_1 + (j − i_h)] then
            {output {(i_1, d_1), (i_2, d_2), ..., (i_{h−1}, d_{h−1})};
            F ← F − {(i_1, d_1), (i_2, d_2), ..., (i_{h−1}, d_{h−1})};
            goto NEXTj};
NEXTj    k ← |F|
    }
output F.
```

Apart from the redefinition of F_j, the main novelty in LD2 is the adjustment made to replace i_k by j in the case that $d_k = j - i_k$, thus ensuring that the leftmost and rightmost only of the identical subwords in a repetition are stored in F_j. For example, given the word

$$w = abababacabababacabababa,$$

the output of LD2 would be

$$F_{23} = \{(1,0),(17,8),(23,2)\},$$

encoding the starting positions $1, 9, 17, 19, 21, 23$. In general, starting positions corresponding to an output set (2.2) are determined by the following rule:

```
for h ← 1 to k do
    if d_h = 0 then
        i_h is a starting position of a Lyndon word
    else
        {i_{h−1} + d_h, i_{h−1} + 2d_h, ..., i_{h−1} + rd_h} are starting positions;
```

where $r = (i_h - i_{h-1})/d_h$. Observe that, in view of Lemma 2.4, the size of any single output produced by LD2 is $O(\log n)$. In fact, in certain cases, the compression of the output can be even more pronounced: the output corresponding to $w = a^n$ is $F = \{(1,0),(n,1)\}$, for example. In conjunction with Lemmas 2.2 and 2.3, Lemma 2.4 also gives rise to the main result of this section, which we now formally state:

Theorem 2.1 Algorithm LD2 computes the starting positions of the Lyndon words of a given word w of length n in $O(n \log n)$ time and $O(\log n)$ space. $\diamondsuit$

3 SOME ASYMPTOTIC RESULTS

In this section we investigate the asymptotic behaviour of Algorithm LD2. We demonstrate that, over a sufficiently large alphabet or, alternatively, over the 1-decomposable words on any alphabet, Algorithms LD1 and LD2 require on average at most

$$n + \log_2 n - 2$$

letter tests for their execution. For the decomposition problem, this is approximately the same as the number of tests required by Duval's algorithm [D83]; as we shall see in the next section, however, when applied to the canonization problem, Algorithm LD2 has a significant advantage.

Theorem 3.1 For a given integer $\alpha > 0$ and a given real number $x > 0$, let $A^{+}_{\alpha,x}$ denote the set of all words W of length $n = \lceil \alpha x \rceil$ whose letters are drawn from an alphabet of size α. Let $D^{1}_{\alpha,x}$ denote the set of all words of $A^{+}_{\alpha,x}$ which are 1-decomposable. Then

(a) $f_{\alpha,x} \equiv |D^{1}_{\alpha,x}|/|A^{+}_{\alpha,x}| = (1 - 1/\alpha)^{n-1}$;

(b) $\lim_{\alpha\to\infty} f_{\alpha,x} = (1/e)^{x}$;

(c) the average number of Lyndon words in a word $W \in D^{1}_{\alpha,x}$ is at most

$$\log_2(\min\{\alpha, n\} + 1).$$

Proof By hypothesis, $A^{+}_{\alpha,x}$ contains α^n distinct words. Observe that if every $W[1,j]$, $1 \le j \le n$, contains a single unique minimum letter, then $W[1]$ may assume any one of α values, but that thereafter, for every integer $i \in [2,n]$, $W[i]$ may assume every value except the current minimum letter of the subword $W[1,i-1]$. Thus $|D^{1}_{\alpha,x}| = \alpha(\alpha-1)^{n-1}$ and

$$f_{\alpha,x} = \alpha(\alpha-1)^{n-1}/\alpha^n,$$

from which (a) follows; (b) then follows from elementary calculus and the fact that

$$(1-1/\alpha)^{\alpha x - 1} \le (1-1/\alpha)^{\lceil \alpha x \rceil - 1} \le (1-1/\alpha)^{\alpha x}.$$

To prove (c), let us first replace each letter of A by its *rank* — that is, by a positive integer in the range $[1,\alpha]$ denoting the letter's position in the alphabet. $W = W[1,n]$ then becomes a string on these integers. Given an integer $j \in [1,n]$, let k denote the (unique) minimum integer in $W[1,j-1]$, where for $j = 1$ we take $k \equiv \alpha + 1$. Then the expected value of the next minimum integer to occur in $W[j,n]$ is $k/2$, so that the expected sequence of minima is represented by $(\alpha+1)/2, (\alpha+1)/4, \ldots$. Each of these minima in fact marks the first position of a Lyndon word of W, and there will be $\log_2(\alpha+1)$ of them.

The expected position of the next minimum integer may be determined as follows. Let

$$p_k \equiv \Pr\{W[j] \ge k\} = (\alpha - k + 1)/\alpha,$$

and let n_k denote the expected number of positions $j, j+1, \ldots, j+h$ until the first value $W[j+h] < k$ occurs. Then for $j = 1$, $n_k = p_k = 0$, while for $j > 1$ ($p_k > 0$),

$$\begin{aligned} n_k &= (1-p_k) + 2(1-p_k)p_k + \cdots + k(1-p_k)p_k^{k-1} + \cdots \\ &= (1-p_k)(1 + 2p_k + 3p_k^2 + \cdots) \\ &= 1/(1-p_k) \\ &= \alpha/(k-1). \end{aligned}$$

Thus the expected gaps between the minima are represented by $1, 2\alpha/(\alpha+1), 4\alpha/(\alpha+1), \ldots$; for a string of length n, there will be $\log_2(n+1)$ such gaps after the starting position $j = 1$. The result follows. $\diamondsuit$

We remark that a result similar to Theorem 3.1(c) holds for arbitrary words W; that is, in general, the expected number of Lyndon words is logarithmic in the minimum of word length and alphabet size.

From Theorem 3.1(a)-(b) we see that, for cases in which alphabet size α is large with respect to word size n, x is small, so that $f_{\alpha,x}$ is close to one. Indeed, for any fixed word length n and any real number $\epsilon > 0$, there exists an integer α^* such that for every alphabet size $\alpha > \alpha^*$, $1 - f_{\alpha,x} < \epsilon$. Hence, by a sufficiently small choice of ϵ — that is, for sufficiently large alphabet size α — it follows from Theorem 3.1(c) that $L_{\alpha,x}$, the expected number of Lyndon words in a word $W \in A^{+}_{\alpha,x}$, is at most $\log_2(n+1)$. We have

Theorem 3.2 Over all words W of length n defined on a sufficiently large alphabet, or over all 1-decomposable words W defined on any alphabet, Algorithms LD1 and LD2 execute on average at most $n + \log_2 n - 2$ tests on the letters of W.

Proof As already remarked about LD1, the number of letter tests required for a 1-decomposable word is $n+m-2$, where m is the number of Lyndon words found. $\diamondsuit$

On smaller alphabets, or on words which are not 1-decomposable, the average case behaviour of Algorithm LD2 appears to be much more difficult to analyze. So far, for these other cases, there is only experimental evidence of the relative efficiency of LD2 compared to other algorithms — this matter is discussed further in Section 4 in the context of the circular word canonization problem. We conclude this section with one further asymptotic result, related to another special case: we show that words (or subwords) which are repetitions are an asymptotically rare phenomenon. Since, as we have seen, repetitions — especially nested repetitions — are one main source of additional tests in LD2, this result provides further justification for considering LD2 in preference to other algorithms.

We consider the set A_n^+ of all words of length n on an alphabet A of size $\alpha = |A|$. Further, for any given integer $k \geq 1$, let $A_{n,k}^+$ denote the subset of A_n^+ consisting of all words w of length n such that $w = u^k$. Then for $k > 1$ and $A_{n,k}^+ \neq \emptyset$, every word in $A_{n,k}^+$ is a repetition, which we say is of *multiplicity* k. Observe that for distinct integers k and k', $A_{n,k}^+$ and $A_{n,k'}^+$ are not necessarily disjoint; for example, $w = (ab)^6$ occurs in $A_{12,2}^+$ and $A_{12,3}^+$ as well as in $A_{12,6}^+$. Suppose that the word length is expressed in the form

$$n = p_1^{q_1} p_2^{q_2} \cdots p_r^{q_r}, \qquad \ldots(3.3)$$

where the p_j, $1 \leq j \leq r$, are distinct primes and $1 < p_1 < p_2 < \cdots < p_r$. We make the following remarks:

* $|A_n^+| = \alpha^n$.
* $|A_{n,k}^+| = \alpha^{n/k}$, if $k \mid n$;
  $= 0$, otherwise.
* For every prime number p and every positive integer h, $A_{n,hp}^+ \subseteq A_{n,p}^+$.
* For every pair of integers $k \mid n$ and $k' \mid n$,
  $$A_{n,k}^+ \cap A_{n,k'}^+ = A_{n,\delta}^+,$$
  where $\delta = \gcd\{n/k, n/k'\}$; in particular, when $\delta = 1$,
  $$|A_{n,k}^+ \cap A_{n,k'}^+| = \alpha.$$

We have then

Theorem 3.3 For $\alpha > 1$, let $f_{\alpha,n}^*$ denote the fraction of words of A_n^+ which are repetitions. Then for n given by (3.3),

$$\begin{aligned} f_{\alpha,n}^* &= \{\sum_{1\leq j\leq r} \alpha^{n/p_j} - \alpha\}/\alpha^n, && \text{if } r > 1; \\ &= \alpha^{-n(p_1-1)/p_1}, && \text{if } r = 1. \ \diamondsuit \end{aligned}$$

$f_{\alpha,n}^*$ is maximized for given n by choosing $\alpha = 2$; for given α, $f_{\alpha,n}^*$ generally decreases with n, but locally tends to assume larger values when n is a product of small primes. The first few values are as follows:

$$f_{\alpha,2}^* = 1/\alpha;\ f_{\alpha,3}^* = f_{\alpha,4}^* = 1/\alpha^2;\ f_{\alpha,5}^* = f_{\alpha,8}^* = 1/\alpha^4;$$

$$f_{\alpha,7}^* = f_{\alpha,9}^* = 1/\alpha^6; f_{\alpha,6}^* = (\alpha^3 + \alpha^2 - \alpha)/\alpha^6 < (\alpha+1)/\alpha^4;$$

$$f_{\alpha,10}^* = (\alpha^5 + \alpha^2 - \alpha)/\alpha^{10} < (\alpha^3+1)/\alpha^8.$$

More generally, it is straightforward to show that

$$f_{\alpha,n}^* \leq r/\alpha^{n/2} < (\log_{10} n + 2)/2^{n/2}.$$

Thus, even for $\alpha = 2$, long words (or subwords) are very rarely repetitions. In view of Lemma 2.4, we remark that the words which lead to the worst case behaviour of Algorithm LD2 contain long repetitive subwords which are a small subset of the repetitions counted here. Thus the worst case behaviour of LD2 is an asymptotically rare event.

4 EFFICIENCY COMPARISONS

In this section we introduce a new problem, the circular word canonization problem, and show that it is solved by any algorithm which computes the Lyndon decomposition of a word. Thus, in particular, both LD2 and Duval's algorithm, mentioned in the Introduction, apply to both problems. However, there exist two other algorithms, due to Booth and to Shiloach, which compute the canonical form of a circular word but which are not known to have variants that handle Lyndon decomposition. In this section, then, we compare all four algorithms as they apply to the circular word canonization problem. The canonization version of LD2 will be called, for a reason which appears below, FMSP.

Given a word W, let $X = W^2$. Then the *canonization problem* on W is to determine those integer values of $i \in [1, n]$ for which $X[i, i+n-1]$ is lexicographically least. (Here we state the problem in terms of W^2 merely to avoid introducing arithmetic modulo n; all the algorithms discussed here actually work on W rather than X.) Such values of i will be called, following Shiloach [S81], *minimum starting points* (MSPs). Let us say that W has *index* k if W contains k MSPs but not $k+1$. Then the following lemma is easily proved by induction on k:

Lemma 4.1 A word W of length n has index $k > 1$ if and only if k is the largest integer for which W is a repetition of multiplicity k. $\diamondsuit$

It follows from this result that a complete solution of the canonization problem can always be achieved by specifying the smallest MSP i together with the index k; then the complete set of MSPs is just

$$\{i, i+d, \ldots, i+(k-1)d\},$$

where the integer $d = n/k$ is called the *offset*. Thus, determining MSPs for each of two given words W_1 and W_2 provides us with a means of determining whether or not W_2 is a cyclic permutation of W_1; it is this observation which relates the MSP computation to the problems of computational geometry [AT78] and graph theory [CB81] mentioned in the Introduction.

Before studying the particular algorithms, we first prove a result which makes clear that any Lyndon decomposition algorithm is essentially also a canonization algorithm:

Theorem 4.1 Suppose that a word W has Lyndon decomposition $u_1^{k_1} u_2^{k_2} \ldots u_m^{k_m}$, where $m \geq 1$ and $u_1 > u_2 > \cdots > u_m$. For every integer $j \in [1, m]$, let i_j denote the starting position of the subword $u_j^{k_j}$. Then i_{m-1} is an MSP of W if and only if all of the following conditions simultaneously hold:

(a) $m > 1$;
(b) $k_m = 1$;
(c) u_m is a prefix of u_{m-1};
(d) $u_m u_1^{k_1} \ldots u_{m-2}^{k_{m-2}} \geq u_{m-1}$.

Otherwise, i_m is an MSP of W.

Proof We first consider each of the conditions in turn, showing that if each one of them independently is false, then i_m is an MSP and i_{m-1} is not. This is trivially so for (a), since if $m \not> 1$, then $m = 1$. In case (b), if $k_m > 1$, then $W[i_m, n] = u_m^{k_m}$ is a minimum subword of length $k_m|u_m|$; if $m > 1$, $W[i_m, n]$ is the only such minimum subword. Observe now that condition (d) implies condition (c). Hence if (c) is false, then (d) also is false, so that i_m is an MSP while i_{m-1} is not.

Now suppose that conditions (a)-(d) all hold. Then i_m can be an MSP only if equality holds in condition (d), and in this case i_{m-1} also is an MSP; otherwise, if $u_m u_1^{k_1} \ldots u_{m-2}^{k_{m-2}} > u_{m-1}$, it follows that i_{m-1} is the unique MSP. $\diamondsuit$

Theorem 4.1 provides us with a methodology for specifying an MSP of W (either i_{m-1} or i_m). In the case that i_{m-1} and i_m are both MSPs, the offset may be computed from

$$d = (i_m - i_{m-1})/k_{m-1},$$

thus determining k and a complete set of MSPs. Since conditions (a)-(d) of Theorem 4.1 can all be evaluated in $O(n)$ time and constant space, it follows that every Lyndon decomposition algorithm converts into a corresponding canonization algorithm with unchanged asymptotic time and space complexity.

As noted above, three algorithms for the canonization problem have already been published. These algorithms are strikingly different: the first [B80] is a variant of the failure function algorithm [KMP77]; a second [S81] is based on a sieve technique which eliminates positions in W that cannot be MSPs; and the third [D83] is as we have seen essentially a Lyndon decomposition algorithm. Table 4.1 compares the four known canonization algorithms in terms of their use of space and time. The space measure given is in addition to the storage required for the original word W. The time measure used is the number of "tests" performed on letters of W: a *test* is a binary function which returns one of $\{=, \neq\}$ or one of $\{<, \geq\}$ or one of $\{>, \leq\}$. (The time measure originally used for the three previously published algorithms was the number of ternary *comparisons* returning one of $\{<, =, >\}$; on a binary computer, tests seem more appropriate.) For FMSP the bounds on the number of tests are identical to those for LD2; for Duval's algorithm, however, the application to the canonization problem depends on a "proposition" [D83, p. 380] whose effect is to double to $4n$ the upper bound on the number of tests.

Comparison of Four Canonization Algorithms

measure	*Booth*	*Shiloach*	*Duval**	*FMSP*
space	$\Theta(n)$	$\Theta(1)$	$\Theta(1)$	$O(\log n)$
no. of tests	$\geq 3n-3$ $\leq 6n$	$\geq n-1$ $\leq 2n-2$	$\geq n-1$ $\leq 4n$	$\geq n-1$ $O(n \log n)$

* Another implementation of Duval's algorithm requires $\Theta(n)$ space, but no more than $3n$ tests.

Table 4.1

Although the measures of Table 4.1 seem to favour Shiloach's and Duval's algorithms, nevertheless we have already seen, in Section 3, that FMSP executes very efficiently on 1-decomposable words and on words whose length is small with respect to alphabet size. In fact we shall see that, for such words, FMSP requires considerably fewer letter tests than Duval's algorithm and, over all words, requires substantially less time than Shiloach's algorithm.

We turn first to a consideration of Duval's algorithm [D83]; in particular, its execution on a 1-decomposable word W of length n which contains m Lyndon words. An inspection of the algorithm easily reveals that, for each position after the first which marks a Lyndon word, two letter tests must be made; also that, for each position which does not mark a Lyndon word, one letter test is made. Thus, at least $n+m-2$ tests are required by Duval's algorithm, the same as the total for Algorithm LD2. However, it is not difficult to see that Duval's algorithm must also perform an additional $i_m - 1$ letter tests (until, as pointed out in [D83], $j-k$ "first equals" $n+1$), where i_m is the position which marks the m^{th} Lyndon word of W. On the other hand, for 1-decomposable words, the set F formed by the execution of FMSP remains empty, and it follows that no additional letter tests are required beyond those required by LD2. Then, denoting by t_D and t_{IS} the number of tests on W performed by Duval's algorithm and FMSP, respectively, we find that

$$t_D - t_{IS} = i_m - 1 \geq m - 1.$$

As we have seen in the previous section, the expected value of i_m is close to n and is thus in $\Omega(n)$. Hence

Theorem 4.2 Over all circular words of length n defined on a sufficiently large alphabet, or over all 1-decomposable circular words defined on any alphabet, Algorithm FMSP requires on average $\Theta(n)$ fewer letter tests than are required by Duval's algorithm; moreover, for any particular 1-decomposable word W with m Lyndon words, the number of letter tests required by Algorithm FMSP is less by at least $m-1$ than that required by Duval's algorithm. $\diamondsuit$

We remark also that, for 1-decomposable words, the number of letter tests required by Shiloach's algorithm is equal to that required by FMSP only if $m = n$; otherwise, the Shiloach algorithm requires more tests.

In the absence of theoretical results for k-decomposable words, $k > 1$, we present here the average results of experiments on large numbers of words of given length n defined on alphabets of size α. These words were randomly generated with all α^n words equally likely. The four algorithms were implemented in a C program which compares their outputs (MSP and k) and which records both CPU time and number of letter tests for each word processed. Each algorithm was compiled using two different "optimizing" compilers, and for each algorithm its fastest implementation was chosen. Typical results are shown in Tables 4.2 and 4.3. (The implementations of Booth's and Shiloach's algorithms include correction of errors found in the original published versions.)

Average Number of Tests for Four Canonization Algorithms

α	n	*Booth*	*Shiloach*	*Duval*	*FMSP*
50	100	442	197	154	108
	10	41	18	16	11
	5	18	8	8	5
25	36	167	69	62	42
4	100	471	174	192	161
	36	170	62	75	60

Table 4.2

Average CPU Time* for Four Canonization Algorithms

α	n	*Booth*	*Shiloach*	*Duval*	*FMSP*
50	100	148.7	75.3	18.6	11.6
	10	8.1	8.7	2.2	1.6
	5	4.1	5.1	1.3	1.0
25	36	31.8	28.5	7.6	4.9
4	100	95.4	69.9	20.6	19.2
	36	31.8	26.5	7.6	6.8

* Normalized according to the FMSP time for $\alpha = 50$, $n = 5$.

Table 4.3

We make the following remarks about the experimental results displayed in Tables 4.2 and 4.3:

* FMSP requires fewer tests and less CPU time in all cases shown. Its advantage over other algorithms is more pronounced for large alphabeting, reflects the greater incidence in small alphabets of subwords which are near-repetitions.
* The Duval algorithm consistently requires much less CPU time than the Shiloach algorithm, even though in certain cases (especially for smaller alphabet size) it requires more tests. This illustrates the danger of relying on an artificial measure such as tests to gauge the efficiency of algorithms; of course the reason for the discrepancy is that, at least in certain instances, the housekeeping required in the Duval algorithm is much simpler and less time-consuming than that of the Shiloach algorithm.

The picture changes considerably when the algorithms are executed on words which consist of $k > 1$ identical subwords (Table 4.4). As discussed in Section 3, these are cases which are difficult for FMSP, and they are also cases which Duval's algorithm handles very efficiently. As we have seen, even for $\alpha = 2$, they are rare.

Average CPU Time* for Multiple Subwords

α	n	k	*Booth*	*Shiloach*	*Duval*	*FMSP*

25	36	2	43.4	25.6	5.2	7.5
		3	44.2	38.2	5.9	10.4
		12	43.5	28.5	4.4	9.1
		18	43.4	29.8	4.5	7.9
4	36	2	43.2	25.3	5.9	10.4
		3	42.9	37.7	5.8	10.6
		12	44.3	28.5	4.8	10.2
		18	43.3	29.3	4.6	7.8
1	36	36	43.9	28.4	4.6	6.8

* Normalized as in Table 4.3.

Table 4.4

APPENDIX A

Here we outline a proof of Lemma 2.4; that is, for a given integer $k \geq 2$, we find the least integer $j = j_k$ such that $W[1,j]$ induces exactly k entries in the set F_j, where F_j is constructed according to the strategy of Algorithm LD2.

It is clear that $j_2 = 2$, since $W[1,2] = aa$ yields $F_2 = \{1,2\}$. For $k = 3$ two strategies are possible. First, we may try to use the shortest possible repetition as a prefix to $W[1,j]$; this leads to $W[1,5] = ababa$ with a corresponding $F_5 = \{1,3,5\}$. Hence $j_3 \leq 5$. Alternatively, we may seek to prepend to the solution for $k = 2$ a minimum length prefix u which adds an additional element to F_j; by Lemma 2.3(b), $|u| > j_2 = 2$. We find that we can form $W[1,5] = aabaa$ corresponding to $F_5 = \{1,4,5\}$. Since there is no other strategy possible, we conclude that $j_3 = 5$.

Now consider $k = 4$. If we prepend a word of length $j_3 + 1$ to either one of the solutions for $k = 3$, we find $j_4 \leq 2j_3 + 1 = 11$. (Prepending to $ababa$ yields, for example, $W[1,11] = ababacababa$ with $F_{11} = \{1,7,9,11\}$, while prepending to $aabaa$ yields $W[1,11] = aabaabaabaa$ or $aabaacaabaa$ with $F_{11} = \{1,7,10,11\}$.) But in this case we can do better: since $aabaa$ has a prefix aab which is not duplicated, we can prepend that prefix, yielding $W[1,8] = aabaabaa$, for which $F_8 = \{1,4,7,8\}$. Observe that since the word $aabaa$ is of minimum length, so also by construction is its prefix aab; thus no other word formed using this strategy could be shorter. Observe also that as for $k = 3$ no other strategy is available to us: either the minimum length word for $k = 4$ must have a duplicated prefix or a non-duplicated prefix whose length is at least $j_3 + 1$. Hence $j_4 = 8$.

Similarly, for $k = 5$, we can form

$$W[1,17] = W[1,8]cW[1,8] = (aabaabaa)c(aabaabaa)$$

by introducing a new letter c, or

$$W[1,17] = (aabaac)(aabaac)aabaa$$

by duplicating the prefix of $W[1,11]$. These solutions yield $F_{17} = \{1,10,13,16,17\}$ or $\{1,7,13,16,17\}$, respectively. Thus $j_5 = 17$.

We can imagine proceeding in this way, applying two strategies at each step to derive j_{k+1} from j_k. Table A.1 shows the results of this procedure for the first few values of k: j_k is the word length obtained by duplicating the prefix, while for $k \geq 3$, j'_k results from prepending j_{k-1} together with a new letter.

Minimum Word Lengths Corresponding to $k = |F_j|$

k	j_k	j'_k
2	2	2
3	5	5

4	8	11
5	17	17
6	26	35
7	53	53
8	80	107
9	161	161
10	242	323

Table A.1

For $k \geq 3$, we can express these strategies formally as follows:

$$\begin{aligned} j_k &= j'_{k-1} + \lceil j'_{k-1}/2 \rceil, \text{ if } k \text{ is even;} \\ &= 2j_{k-1} + 1, \qquad\quad \text{if } k \text{ is odd.} \end{aligned}$$

Since $j_2 = j'_2 = 2$, it is straightforward to show from these recurrence relations that for every odd k, $j_k = j'_k$. Moreover, for every even $k \geq 4$,

$$\begin{aligned} j_k &= (2j_{k-2} + 1) + \lceil (2j_{k-2} + 1)/2 \rceil \\ &= 3j_{k-2} + 2. \end{aligned}$$

Since $j_2 = 2$, these equations yield the solution

$$j_k = 3^{k/2} - 1. \qquad \ldots (A.1)$$

Then for every even k, (A.1) gives the minimum length subword $W[1, j]$ for which $|F_j| = k$. As we have seen, this minimum is achieved. Moreover, for odd k, $j_k > 3^{k/2} - 1$, and Lemma 2.4 follows.

Finally, we observe that the minimum values j_k are dependent on the alphabet size; in particular, for smaller alphabets, the j_k will tend to become larger for larger values of k. Thus, for smaller alphabets, the upper bound expressed by Lemma 2.4 will become smaller.

REFERENCES

[AHU74] A. V. Aho, J. E. Hopcroft & J. D. Ullman, *The Design and Analysis of Computer Algorithms*, Addison-Wesley (1974).

[AT78] A. G. Akl & T. G. Toussaint, **An improved algorithm to check for polygon similarity**, *Inform. Process. Lett. 7-3* (1978) 127-128.

[B80] K. S. Booth, **Lexicographically least circular substrings**, *Inform. Process. Lett. 10-4&5* (1980) 240-242.

[CB81] C. J. Colbourn & K. S. Booth, **Linear time isomorphism algorithms for trees, interval graphs, and planar graphs**, *SIAM J. Computing 10* (1981) 203-225.

[CFL58] K. T. Chen, R. H. Fox & R. C. Lyndon, **Free differential calculus IV**, *Ann. of Math. 68* (1958) 81-95.

[D83] J. P. Duval, **Factoring words over an ordered alphabet**, *J. Algorithms 4* (1983) 363-381.

[IS89] C. S. Iliopoulos & W. F. Smyth, **PRAM algorithms for identifying polygon similarity**, *Springer-Verlag Lecture Notes in Computer Science 401*, H. Djidjev (ed.) (1989) 25-32.

[KMP77] D. E. Knuth, J. Morris & V. Pratt, **Fast pattern matching in strings**, *SIAM J. Computing 6-2* (1977) 323-350.

[S81] Y. Shiloach, **Fast canonization of circular strings**, *J. Algorithms 2* (1981) 107-121.

ACKNOWLEDGEMENTS

The work of the second author was supported in part by Grant No. A8180 of the Natural Sciences & Engineering Research Council of Canada, and by Grant No. GR/E 75752 of the Science & Engineering Research Council of the UK. The authors express their appreciation of the fine work of Kerstin Baxter in programming the canonization algorithms.