Journal of Discrete Algorithms 28 (2014) 41–48

Contents lists available at ScienceDirect

Journal of Discrete Algorithms

www.elsevier.com/locate/jda

A subquadratic algorithm for minimum palindromic factorization

Gabriele Fici [a], Travis Gagie [b], Juha Kärkkäinen [b,*], Dominik Kempa [b]

[a] Dipartimento di Matematica e Informatica, Università di Palermo, Italy
[b] Department of Computer Science, University of Helsinki, Finland

ARTICLE INFO

Article history:
Available online 6 August 2014

Keywords:
String algorithms
Palindromes
Factorization

ABSTRACT

We give an $\mathcal{O}(n \log n)$-time, $\mathcal{O}(n)$-space algorithm for factoring a string into the minimum number of palindromic substrings. That is, given a string $S[1..n]$, in $\mathcal{O}(n \log n)$ time our algorithm returns the minimum number of palindromes $S_1, \ldots, S_\ell$ such that $S = S_1 \cdots S_\ell$. We also show that the time complexity is $\mathcal{O}(n)$ on average and $\Omega(n \log n)$ in the worst case. The last result is based on a characterization of the palindromic structure of Zimin words.

1. Introduction

Palindromic substrings are a well-studied topic in stringology and combinatorics on words. Since a single character is a palindrome, there are always between n and $\binom{n}{2} + n = \Theta(n^2)$ non-empty palindromic substrings in a string of length n. There are only $2n - 1$ possible centers of those substrings, however – i.e., the n individual characters and the $n - 1$ gaps between them – so many algorithms involving palindromic substrings still run in subquadratic time. For example, Manacher [13] gave a linear-time algorithm for listing all the palindromic prefixes of a string. Apostolico, Breslauer and Galil [3] observed that Manacher's algorithm can be used to list in linear time all maximal palindromic substrings, which are those that cannot be extended without changing the position of the center. Other linear-time algorithms for this problem were given by Jeuring [10] and Gusfield [8]. Since any palindromic substring is contained within the maximal palindromic substring with the same center, the list of all maximal palindromic substrings can be viewed as a linear-space representation of all palindromic substrings. For more discussion of algorithms involving palindromes, we refer the reader to Jeuring's recent survey [11].

Palindromes are a useful tool for investigating string complexity; see, e.g., [2]. A natural measure of the asymmetry of a string S is its palindromic length $\mathsf{PL}(S)$, which is the minimum number of palindromic substrings into which S can be factored. That is, $\mathsf{PL}(S)$ is the minimum number ℓ such that there exist palindromes $S_1, \ldots, S_\ell$ whose concatenation $S_1 \cdots S_\ell = S$. For example, $\mathsf{PL}(abaab) = 2$ and $\mathsf{PL}(abaca) = 3$. Notice that, since a single character is a palindrome, $\mathsf{PL}(S)$ is always well-defined and lies between 0 and $|S|$, or 1 and $|S|$ if S is non-empty. In fact, $\mathsf{PL}(S[1..i]) - 1 \le \mathsf{PL}(S[1..i+1]) \le \mathsf{PL}(S[1..i]) + 1$ for $i < |S|$: first, if $S_1, \ldots, S_{\ell-1}, S[h..i+1]$ is a factorization of $S[1..i+1]$ into ℓ palindromic substrings, then $S_1, \ldots, S_{\ell-1}, S[h], S[h+1..i]$ is a factorization of $S[1..i]$ into $\ell + 1$ palindromic substrings; second, if $S_1, \ldots, S_\ell$ is a

* Corresponding author.
E-mail addresses: gabriele.fici@unipa.it (G. Fici), travis.gagie@cs.helsinki.fi (T. Gagie), juha.karkkainen@cs.helsinki.fi (J. Kärkkäinen), dominik.kempa@cs.helsinki.fi (D. Kempa).

http://dx.doi.org/10.1016/j.jda.2014.08.001
1570-8667/© 2014 Published by Elsevier B.V.

```
Algorithm Palindromic-length(S[1..n])
 1: PL[0] ← 0
 2: P ← ∅
 3: for j ← 1 to n do
 4:     P′ ← ∅
 5:     foreach i ∈ P do
 6:         if i > 1 and S[i − 1] = S[j] then
 7:             P′ ← P′ ∪ {i − 1}
 8:     if j > 1 and S[j − 1] = S[j] then
 9:         P′ ← P′ ∪ {j − 1}
10:     P ← P′ ∪ {j}
11:     PL[j] ← j
12:     foreach i ∈ P do
13:         PL[j] ← min(PL[j], PL[i − 1] + 1)
14: return PL[n]
```

Fig. 1. A simple quadratic-time algorithm for computing the palindromic length. Every iteration of the for loop in line 3 starts with $P = P_{j-1}$ and ends with $P = P_j$.

factorization of $S[1..i]$ into ℓ palindromic substrings, then $S_1, \dots, S_\ell, S[i+1]$ is a factorization of $S[1..i+1]$ into $\ell + 1$ palindromic substrings.

We became interested in palindromic length because of a recent conjecture by Frid, Puzynina and Zamboni [7]. Some infinite strings (e.g., the regular paperfolding sequence) are highly asymmetric in that they contain only a finite number of distinct palindromic substrings; see [6] for more discussion. For such strings, the palindromic length of any finite substring is proportional to that substring's length. In contrast, for other infinite strings (e.g., the infinite power of any palindrome), the palindromic length of any finite substring is bounded. Frid et al. conjectured that all such infinite strings are (ultimately) periodic.

It is easy to compute $\mathsf{PL}(S)$ in quadratic time via dynamic programming. Alatabbi, Iliopoulos and Rahman [1] recently gave a linear-time algorithm for computing a minimum factorization of S into *maximal* palindromic substrings, when such a factorization exists; it does not exist for, e.g., *abaca*. Even when such a factorization exists, it may consist of more than $\mathsf{PL}(S)$ substrings; e.g., *abbaabaabbba* can be factored into *abba*, *aba* and *abbba* but cannot be factored into fewer than four maximal palindromic substrings.

In this paper, we give an $\mathcal{O}(n \log n)$-time and $\mathcal{O}(n)$-space algorithm for factoring S into $\mathsf{PL}(S)$ palindromic substrings. The average case time complexity is in fact linear, but the worst case is $\Theta(n \log n)$, which we show by an analysis of the palindromic structure of Zimin words [4, Chapter 5.4].

Independently of us, I, Sugimoto, Inenaga, Bannai and Takeda [9] discovered essentially the same algorithm. Also, Kosolobov, Rubinchik and Shur [12] have recently described an algorithm recognizing strings with a given palindromic length. Their result can be used for computing the palindromic length of a string S in $\mathcal{O}(|S| \cdot \mathsf{PL}(S))$ time.[1]

2. A simple quadratic algorithm

We start by describing a simple algorithm for computing $\mathsf{PL}(S)$ in $\mathcal{O}(n^2)$ time and $\mathcal{O}(n)$ space using the observation that, for $1 \le j \le n$,

$$\mathsf{PL}\big(S[1..j]\big) = \min_i \big\{\mathsf{PL}\big(S[1..i-1]\big) + 1 : i \le j,\ S[i..j] \text{ is a palindrome}\big\}.$$

We compute and store an array $\mathsf{PL}[0..n]$, where $\mathsf{PL}[0] = 0$ and $\mathsf{PL}[i] = \mathsf{PL}(S[1..i])$ for $i \ge 1$. At each step j, we compute the set P_j of the starting positions of all palindromes ending at j from the set P_{j-1} using the observation that $S[i..j]$, $i + 1 \le j - 1$, is a palindrome if and only if $S[i+1..j-1]$ is a palindrome and $S[i] = S[j]$. The algorithm is given in Fig. 1.

The space requirement is clearly $\mathcal{O}(n)$. During the jth step of the algorithm, we use time $\mathcal{O}(|P_j| + |P_{j-1}|)$, so for all the steps we use total time proportional to the number of palindromic substrings in S. For most strings the time is linear (see Theorem 11) but the worst case is quadratic, e.g., for $S = a^n$ or $S = (ab)^{n/2}$.

It is straightforward to modify the algorithm so that it produces an actual minimum palindromic factorization of S, without increasing the running time or space by more than a constant factor.

3. Faster computation of palindromes

In this section, we replace the representation P_j of the palindromes ending at j with a more compact representation G_j that needs only $\mathcal{O}(\log j)$ space and can be computed in $\mathcal{O}(\log j)$ time from G_{j-1}. The representation is based on combinatorial properties of palindromes.

A string y is a *border* of a string x if y is both a prefix of x and a suffix of x, and a *proper* border if $y \ne x$. The following easy lemmas establish a connection between borders and palindromes.

[1] Editors' note: we are satisfied that the results of this paper, and those of [9] and [12], have all been achieved independently.

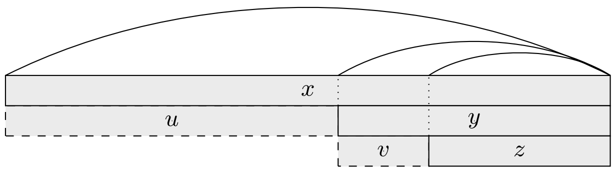


Fig. 2. Setting in Lemma 4.

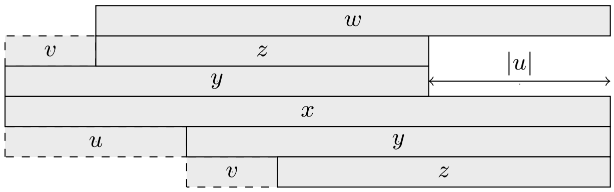


Fig. 3. Proof of Lemma 4(2): if $|u| > |v|$ and $|u| \le |z|$ then w is a palindromic proper suffix of x longer than y.

Lemma 1. *(See [5].) Let y be a suffix of a palindrome x. Then y is a border of x iff y is a palindrome.*

Lemma 2. *(See [5].) Let x be a string with a border y such that $|x| \le 2|y|$. Then x is a palindrome iff y is a palindrome.*

A positive integer $p \le |x|$ is a *period* of a string x if there exists a string w of length p such that x is a factor of w^∞. It is well known that y is a proper border of x if and only if $|x| - |y|$ is a period of x. This, together with Lemma 1, implies the following connection between periods and palindromes.

Lemma 3. *Let y be a proper suffix of a palindrome x. Then $|x| - |y|$ is a period of x iff y is a palindrome. In particular, $|x| - |y|$ is the smallest period of x iff y is the longest palindromic proper suffix of x.*

Now we are ready to state and prove the key combinatorial property of palindromic suffixes.

Lemma 4. *Let x be a palindrome, y the longest palindromic proper suffix of x and z the longest palindromic proper suffix of y. Let u and v be strings such that $x = uy$ and $y = vz$. Then*

(1) $|u| \ge |v|$;
(2) *if* $|u| > |v|$ *then* $|u| > |z|$;
(3) *if* $|u| = |v|$ *then* $u = v$.

Proof. See Fig. 2 for an illustration.

(1) By Lemma 3, $|u| = |x| - |y|$ is the smallest period of x, and $|v| = |y| - |z|$ is the smallest period of y. Since y is a factor of x, either $|u| > |y| > |v|$ or $|u|$ is a period of y too, and thus it cannot be smaller than $|v|$.

(2) By Lemma 1, y is a border of x and thus v is a prefix of x. Let w be a string such that $x = vw$. Then z is a border of w and $|w| = |zu|$, see Fig. 3. Since we assume $|u| > |v|$, we must have $|w| > |y|$. Suppose to the contrary that $|u| \le |z|$. Then $|w| = |zu| \le 2|z|$, and by Lemma 2, w is a palindrome. But this contradicts y being the longest palindromic proper suffix of x.

(3) In the proof of (2) we saw that v is a prefix of x, and so is u by definition. Thus $u = v$ if $|u| = |v|$. □

We will use the above lemma to establish the properties of the set P_j. Let $P_j = \{p_1, p_2, \ldots, p_m\}$ with $p_1 < p_2 < \ldots < p_m$. By *gap* we mean the difference $p_i - p_{i-1}$ of two consecutive values in P_j. The following result has been proven in [14] but we provide a proof for completeness.

Lemma 5. *The sequence of gaps in P_j is non-increasing and there are at most $\mathcal{O}(\log j)$ distinct gaps.*

Proof. For any $i \in [2..m-1]$, if we let $x = S[p_{i-1}..j]$, $y = S[p_i..j]$ and $z = S[p_{i+1}..j]$, we have the situation of Lemma 4 with gaps of $|u|$ and $|v|$. The sequence of gaps is non-increasing by Lemma 4(1). If we have a change of gap, i.e., $|u| > |v|$, we must have $|x| > |u| + |z| > 2|z|$ by Lemma 4(2), i.e., the length of the palindromic suffix is halved in two steps. This cannot happen more than $\mathcal{O}(\log j)$ times. □

We will partition the set P_j by the gaps into $\mathcal{O}(\log j)$ consecutive subsets, each of which can be represented in constant space since it forms an arithmetic progression. For any positive integer Δ, we define $P_{j,\Delta} = \{p_i : 1 < i \le m, p_i - p_{i-1} = \Delta\}$,

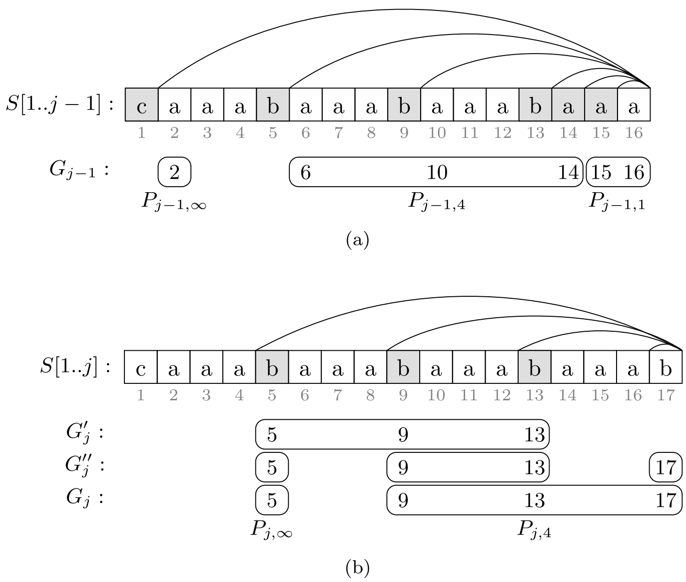


Fig. 4. (a) The palindromic suffixes of $S[1..j-1]$ for $j=17$ start at positions $P_{j-1}=\{2,6,10,14,15,16\}$ and the compact representation is $G_{j-1}=((2,\infty,1),(6,4,3),(15,1,2))$. The shaded symbols will be compared with the next symbol appended to the text. (b) The palindromic suffixes after appending $S[j]$. The sequence G'_j is obtained by taking each triple $(i,\Delta,k)\in G_{j-1}$ and either removing it or replacing it with $(i-1,\Delta,k)$. The resulting sequence $G'_j=((5,4,3))$, however, is no longer a valid gap partitioning because the gap of the first element encoded by triple $(5,4,3)$ is ∞. This is fixed by separating this element into its own triple. At this point we also add the palindromes of length at most 2 to obtain $G''_j=((5,\infty,1),(9,4,2),(17,4,1))$. Finally, we merge neighboring triples with the same Δ to obtain $G_j=((5,\infty,1),(9,4,3))$.

and $P_{j,\infty}=\{p_1\}$. Each non-empty $P_{j,\Delta}$ is represented by the triple $(\min P_{j,\Delta},\Delta,|P_{j,\Delta}|)$. Let G_j be the list of such triples in decreasing order of Δ.

The list G_j is a full representation of P_j of size $\mathcal{O}(\log j)$. We will show that G_j can be computed from G_{j-1} in $\mathcal{O}(|G_{j-1}|)$ time. In the quadratic-time algorithm, each element i of P_{j-1} was either eliminated or replaced by $i-1$ in P_j. The following lemma shows that the decision to eliminate or replace can be made simultaneously for all elements of a partition $P_{j-1,\Delta}$. See Fig. 4a for an example.

Lemma 6. *Let p_i and p_{i+1} be two consecutive elements of $P_{j-1,\Delta}$. Then $p_i-1\in P_j$ iff $p_{i+1}-1\in P_j$.*

Proof. By definition, $p_{i+1}-p_i=\Delta$, and the predecessor of p_i in P_j is $p_{i-1}=p_i-\Delta$. Using the definitions from the proof of Lemma 5, we have the situation of Lemma 4(3), which implies that $S[p_i-1]=S[p_{i+1}-1]=c$. Thus, $p_i-1\in P_j$ iff $S[j]=c$ iff $p_{i+1}-1\in P_j$. □

Thus, when computing G_j, each triple $(i,\Delta,k)\in G_{j-1}$ will be either eliminated or replaced by $(i-1,\Delta,k)$. The resulting sequence of triples is

$$G'_j=\{(i-1,\Delta,k):(i,\Delta,k)\in G_{j-1}, i>1, \text{ and } S[i-1]=S[j]\},$$

which is a full representation of all palindromes longer than two in P_j.

However, the triples in G'_j may no longer perfectly correspond to the partitions $P_{j,\Delta}$ because the gaps may have changed. Specifically, if the smallest element p_i in $P_{j-1,\Delta}$ is replaced by p_i-1 but its predecessor $p_{i-1}=p_i-\Delta$ in P_{j-1} is eliminated, then p_i-1 is not in $P_{j,\Delta}$ but it is, at this point, represented by the triple (p_i-1,Δ,k). Note that only the smallest element of each partition can be affected by this. In such cases, we separate the first element into its own triple, i.e., we replace (p_i-1,Δ,k) with $(p_i-1,\Delta',1)$ and (if $k>1$) $(p_i-1+\Delta,\Delta,k-1)$, where Δ' is the new gap preceding p_i-1 in P_j. We will also add separate triples to represent palindromes of lengths one and (possibly) two.

Let G''_j be the sequence of triples obtained from G'_j by the above process (see lines 8–21 in Fig. 8). It represents exactly the palindromes in P_j and the Δ-values are now correct, but there may be multiple triples with the same Δ. Thus we obtain the final sequence G_j from G''_j by merging triples with the same Δ.

The full procedure for computing G_j from G_{j-1} is shown on lines 4–30 in Fig. 8 and the example of computation is given in Fig. 4b. Each triple is processed in constant time and the number of triples never exceeds $\mathcal{O}(|G_{j-1}|)$.

Lemma 7. *G_j can be computed from G_{j-1} in $\mathcal{O}(|G_{j-1}|)=\mathcal{O}(\log j)$ time.*

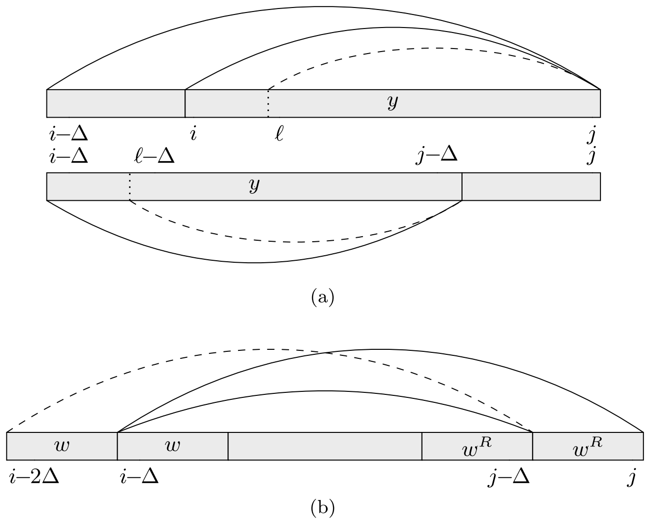


Fig. 5. Proof of Lemma 8. (a) $\ell \in P_j$ iff $\ell - \Delta \in P_{j-\Delta}$ for all $\ell \in [i..j]$. (b) If $i - 2\Delta \in P_{j-\Delta}$ then $S[i-2\Delta..j]$ is a palindrome.

4. Faster factorization

In this section, we will show how to compute $\mathsf{PL}[j]$ from $\mathsf{PL}[0..j-1]$ and G_j in $\mathcal{O}(|G_j|)$ time. The key to fast computation of G_j was the close relation between $P_{j,\Delta}$ and $P_{j-1,\Delta}$. Now we will rely on the relation between $P_{j,\Delta}$ and $P_{j-\Delta,\Delta}$ captured by the following result.

Lemma 8. *If $(i, \Delta, k) \in G_j$ for $k \geq 2$, then $(i, \Delta, k-1) \in G_{j-\Delta}$.*

Proof. By definition, $(i, \Delta, k) \in G_j$ is equivalent to saying that $P_{j,\Delta} = \{i, i+\Delta, \ldots, i+(k-1)\Delta\}$, and we need to show that $P_{j-\Delta,\Delta} = \{i, i+\Delta, \ldots, i+(k-2)\Delta\}$. We will show first that $P_{j-\Delta,\Delta} \cap [i-\Delta+1..j-\Delta] = \{i, i+\Delta, \ldots, i+(k-2)\Delta\}$ and then that $P_{j-\Delta,\Delta} \cap [1..i-\Delta] = \emptyset$.

Since $y = S[i..j]$ and $x = S[i-\Delta..j]$ are palindromes and y is the longest proper border of x, $S[i-\Delta..j-\Delta] = y = S[i..j]$. Thus for all $\ell \in [i..j]$, $\ell \in P_j$ iff $\ell - \Delta \in P_{j-\Delta}$ (see Fig. 5a). In particular, the gaps in both cases are the same and for all $\ell \in [i+1..j]$, $\ell \in P_{j,\Delta}$ iff $\ell - \Delta \in P_{j-\Delta,\Delta}$. Thus $P_{j-\Delta,\Delta} \cap [i-\Delta+1..j-\Delta] = \{i, i+\Delta, \ldots, i+(k-2)\Delta\}$.

We still need to show that $P_{j-\Delta,\Delta} \cap [1..i-\Delta] = \emptyset$, which is true if and only if $i - 2\Delta \notin P_{j-\Delta}$. Suppose to the contrary that $S[i-2\Delta..j-\Delta]$ is a palindrome and let $w = S[i-2\Delta..i-\Delta-1]$. Then $S[j-2\Delta+1..j-\Delta] = w^R$, the reverse of w. Since $z = S[i-\Delta..j-\Delta]$ and $S[i-\Delta..j]$ are palindromes too, we have that $S[i-\Delta..i-1] = w$ and $S[j-\Delta+1..j] = w^R$. Finally, since z is a palindrome, $S[i-2\Delta..j] = wzw^R$ is a palindrome (see Fig. 5b). This implies that $i - 2\Delta \in P_j$ and thus $i - \Delta \in P_{j,\Delta}$, which is a contradiction. □

By the above lemma, $P_{j,\Delta} = P_{j-\Delta,\Delta} \cup \{\max P_{j,\Delta}\}$ whenever $|P_{j,\Delta}| \geq 2$. Thus we can compute $\mathsf{PL}_{j,\Delta} = \min\{\mathsf{PL}[i-1]+1 : i \in P_{j,\Delta}\}$ from $\mathsf{PL}_{j-\Delta,\Delta}$ in constant time. We will store the value $\mathsf{PL}_{j,\Delta}$ in an array $\mathsf{GPL}[1..n]$ at the position $m = \min P_{j,\Delta} - \Delta$. Note that m is the predecessor of $\min P_{j,\Delta}$ in P_j and the position is shared by $\mathsf{PL}_{j-\Delta,\Delta}$ (when $|P_{j,\Delta}| \geq 2$). The following lemma shows that the position is not overwritten by another value between the rounds $j - \Delta$ and j. See Fig. 6 for an example.

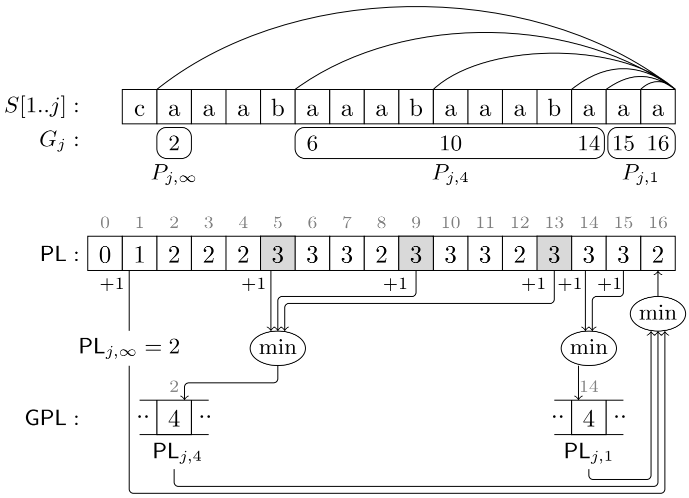
Lemma 9. *Let $m = \min P_{j,\Delta} - \Delta$. For all $\ell \in [j-\Delta+1..j-1]$, $m \notin P_\ell$.*

Proof. Suppose to the contrary that $m \in P_\ell$ for some $\ell \in [j-\Delta+1..j-1]$, i.e., $S[m..\ell]$ is a palindrome. Then $S[m+h..\ell-h]$ for $h = \ell - j + \Delta$ is a palindrome too (see Fig. 7). Since $\ell - h = j - \Delta$ and $m < m+h < m+\Delta = \min P_{j-\Delta,\Delta}$, this contradicts m being the predecessor of $\min P_{j-\Delta,\Delta}$ in $P_{j-\Delta}$. □

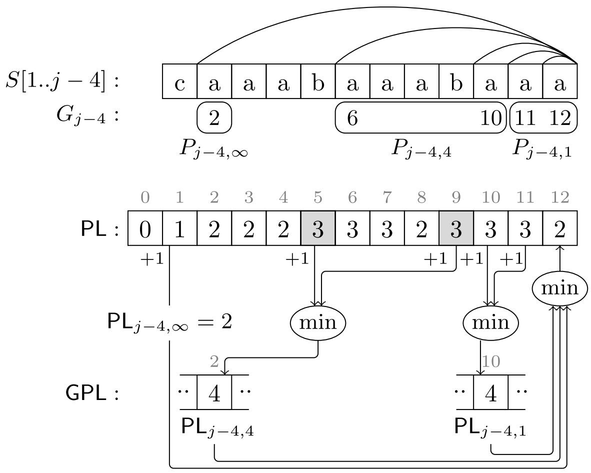
The full algorithm is given in Fig. 8. The running time of round j is $\mathcal{O}(|G_{j-1}| + |G_j|)$. Since $|G_j| = \mathcal{O}(\log j)$ for all j, we obtain the following result.

Theorem 10. *The palindromic length of a string of length n can be computed in $\mathcal{O}(n \log n)$ time and $\mathcal{O}(n)$ space.*

As with the quadratic-time algorithm, the algorithm can be modified to produce an actual minimum palindromic factorization without an asymptotic increase in time or space complexities: we need only store with each palindromic length



(a) Iteration j.



(b) Iteration $j - 4$.

Fig. 6. Example usage of the GPL array for $j = 16$. The value of $\mathsf{PL}_{j,4}$ computed in iteration j depends on shaded elements from PL array. Rather than scanning them all, we apply Lemma 8. Since $|P_{j,4}| \geq 2$ we get $P_{j,4} = P_{j-4,4} \cup \{14\}$. Therefore we can compute $\mathsf{PL}_{j,4}$ as $\min\{\mathsf{PL}_{j-4,4}, \mathsf{PL}[13] + 1\}$. The value of $\mathsf{PL}_{j-4,4}$ was computed during iteration $j - 4$ and stored at position $\min P_{j-4,4} - 4 = \min P_{j,4} - 4 = 2$ in the GPL array, and by Lemma 9 it was not overwritten between iterations $j - 4$ and j. Thus we compute $\mathsf{PL}_{j,4}$ in constant time as $\min\{\mathsf{GPL}[2], \mathsf{PL}[13] + 1\}$ and update GPL[2] with the new value.

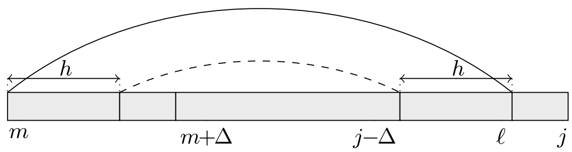


Fig. 7. Proof of Lemma 9: if $m \in P_\ell$ then $m + h \in P_{\ell-h} = P_{j-\Delta}$.

in PL and GPL, the length of the last palindrome in the corresponding minimum factorization. The algorithm is also on-line in the sense that the string is processed from left to right and, for each j, the character $S[j]$ is processed in $\mathcal{O}(\log j)$ time, after which we can report the palindromic length $\mathsf{PL}(S[1..j])$ in constant time and the corresponding factorization in $\mathcal{O}(\mathsf{PL}(S[1..j]))$ time.

5. Average and worst case

In this section, we show that the average case time complexity of the algorithm is linear, but that the worst case is indeed $\Theta(n \log n)$.

```
Algorithm Palindromic-length(S[1..n])
PL[0] ← 0
G ← ()
for j ← 1 to n do
    G' ← ()
    foreach (i, Δ, k) ∈ G do
        if i > 1 and S[i − 1] = S[j] then
            G'.pushback((i − 1, Δ, k))      // appends the given triple
    G'' ← ()
    r ← −j      // makes i − r big enough to act as ∞
    foreach (i, Δ, k) ∈ G' do
        if i − r ≠ Δ then
            G''.pushback((i, i − r, 1))
            if k > 1 then
                G''.pushback((i + Δ, Δ, k − 1))
        else
            G''.pushback((i, Δ, k))
        r ← i + (k − 1)Δ
    if j > 1 and S[j − 1] = S[j] then
        G''.pushback((j − 1, j − 1 − r, 1))
        r ← j − 1
    G''.pushback((j, j − r, 1))
    G ← ()
    (i', Δ', k') ← G''.popfront()      // removes and returns the first triple
    foreach (i, Δ, k) ∈ G'' do
        if Δ' = Δ then
            k' = k' + k
        else
            G.pushback((i', Δ', k'))
            (i', Δ', k') ← (i, Δ, k)
    G.pushback((i', Δ', k'))
    PL[j] ← j
    foreach (i, Δ, k) ∈ G do
        r ← i + (k − 1)Δ
        m ← PL[r − 1] + 1
        if k > 1 then
            m ← min(m, GPL[i − Δ])
        if Δ ≤ i then
            GPL[i − Δ] ← m
        PL[j] ← min(PL[j], m)
return PL[n]
```

Fig. 8. Algorithm for computing the palindromic length in $\mathcal{O}(n \log n)$ time.

Theorem 11. *The average case time complexity of the algorithms in Fig. 1 and in Fig. 8 is $\mathcal{O}(n)$.*

Proof. Consider the set Σ^n of the σ^n strings of length n over an alphabet Σ of size $\sigma > 1$. All of them have a palindromic suffix of length one, σ^{n-1} of them have a palindromic suffix of length two, and the same number have a palindromic suffix of length three (assuming $n \geq 3$). More generally, for $1 \leq k \leq n$, the number of strings with a palindromic suffix of length k is $\sigma^{n-k/2}$ when k is even and $\sigma^{n-(k-1)/2}$ when k is odd. Then the total number of palindromic suffixes in Σ^n is

$$\sum_{i=1}^{\lfloor n/2 \rfloor} \sigma^{n-i} + \sum_{i=1}^{\lceil n/2 \rceil} \sigma^{n-i+1} < \sigma^n/(\sigma-1) + \sigma^{n+1}/(\sigma-1) = \frac{\sigma+1}{\sigma-1}\sigma^n \leq 3\sigma^n.$$

Therefore the average number of palindromes ending at any position is less than three, and both algorithms spend a constant time on average for processing each position. □

We show the worst case complexity of the algorithm by constructing a family of strings based on the Zimin words [4, Chapter 5.4]. Let $Z_0 = \varepsilon$, and $Z_i = Z_{i-1} i Z_{i-1}$ for $i > 0$. The limit of this sequence is the infinite Zimin word $Z = 1213121412131215\ldots$. For a non-negative integer n, let $B(n)$ be the number of 1-bits in the binary representation of n. For example, $B(0) = 0$, $B(1) = 1$, $B(7) = 3$ and $B(8) = 1$.

Lemma 12. *The prefix $Z[1..n]$ of the infinite Zimin word Z has exactly $B(n)$ suffix palindromes.*

Proof. From the definition, it is easy to see that the prefix $Z[1..n]$ has a unique factorization of the form

$$Z[1..n] = Z_{i_k}(i_k + 1) \cdot Z_{i_{k-1}}(i_{k-1} + 1) \cdots Z_{i_2}(i_2 + 1) \cdot Z_{i_1}(i_1 + 1)$$

where $0 \le i_1 < i_2 < \ldots < i_{k-1} < i_k$. For example, $Z[1..10] = Z_3 4 Z_1 2$. Since the length of a factor $Z_i(i+1)$ is 2^i, we must have that $\sum_{j=1}^{k} 2^{i_j} = n$. Thus $i_1, \ldots, i_k$ are the positions of 1-bits in the binary representation of n, and $k = B(n)$.

Let $n_j = 2^{i_j}$ for $j \in [1..k]$. Clearly, $Z[2n_k - n..n]$ is a palindrome of length $2(n - n_k) + 1$ centered at $Z[n_k] = (i_k + 1)$. For example, $Z[6..10] = 21412$ is a palindrome centered at $Z[8] = 4$. Since $Z[n_k]$ is the only occurrence of $(i_k + 1)$ in $Z[1..n]$, there can be no other suffix palindromes with a starting position in $Z[1..n_k]$. By a similar argument, there is exactly one suffix palindrome with a starting position in $Z[n_k + 1..n_k + n_{k-1}]$, the one centered at $Z[n_k + n_{k-1}] = (i_{k-1} + 1)$, and so on. In total, $Z[1..n]$ has exactly k suffix palindromes. □

Theorem 13. *The running time of the algorithm in Fig. 8 for input $Z[1..n]$ is $\Theta(n \log n)$.*

Proof. By Lemma 12, $Z[1..j]$ has exactly $B(j)$ suffix palindromes, i.e., $|P_j| = B(j)$. From the proof it is easy to see that each of the suffix palindromes is at least twice as long as the next shorter suffix palindrome. Thus there are no two identical gaps in P_j and $|G_j| = |P_j| = B(j)$. Since the algorithm spends $\Theta(|G_{j-1}| + |G_j|)$ time in round j, the total time complexity is $\Theta(\sum_{j=1}^{n} B(j))$, which is $\Theta(n \log n)$ [15]. □

Acknowledgements

Many thanks to the organizers and participants of the Stringmasters 2013 workshops in Verona and Prague, and to the anonymous reviewers. This research was partially supported by the Italian MIUR Project PRIN 2010LYA9RH, "Automi e Linguaggi Formali: Aspetti Matematici e Applicativi", and by the Academy of Finland through grant 268324 and grant 118653 (ALGODAN).

References

[1] A. Alatabbi, C.S. Iliopoulos, M.S. Rahman, Maximal palindromic factorization, in: Proceedings of the Prague Stringology Conference, PSC, 2013, pp. 70–77.
[2] J.P. Allouche, M. Baake, J. Cassaigne, D. Damanik, Palindrome complexity, Theor. Comput. Sci. 292 (1) (2003) 9–31.
[3] A. Apostolico, D. Breslauer, Z. Galil, Parallel detection of all palindromes in a string, Theor. Comput. Sci. 141 (1–2) (1995) 163–173.
[4] J. Berstel, A. Lauve, C. Reutenauer, F.V. Saliola, Combinatorics on Words: Christoffel Words and Repetition in Words, CRM Monogr. Ser., vol. 27, American Mathematical Society and Centre de Recherches Mathématiques, 2008.
[5] A. Blondin Massé, S. Brlek, S. Labbé, Palindromic lacunas of the Thue–Morse word, Pure Math. Appl. 19 (2–3) (2008) 39–52.
[6] G. Fici, L.Q. Zamboni, On the least number of palindromes contained in an infinite word, Theor. Comput. Sci. 481 (2013) 1–8.
[7] A.E. Frid, S. Puzynina, L. Zamboni, On palindromic factorization of words, Adv. Appl. Math. 50 (5) (2013) 737–748.
[8] D. Gusfield, Algorithms on Strings, Trees, and Sequences: Computer Science and Computational Biology, Cambridge University Press, 1997.
[9] T. I, S. Sugimoto, S. Inenaga, H. Bannai, M. Takeda, Computing palindromic factorizations and palindromic covers on-line, in: Proceedings of the 25th Symposium on Combinatorial Pattern Matching, CPM, in: Lect. Notes Comput. Sci., vol. 8486, Springer, 2014, pp. 150–161.
[10] J. Jeuring, The derivation of on-line algorithms, with an application to finding palindromes, Algorithmica 11 (2) (1994) 146–184.
[11] J. Jeuring, Finding palindromes: variants and algorithms, in: P. Achten, P. Koopman (Eds.), The Beauty of Functional Code, in: Lect. Notes Comput. Sci., vol. 8106, Springer, 2013, pp. 258–272.
[12] D. Kosolobov, M. Rubinchik, A.M. Shur, Pal^k is linear recognizable online, CoRR, arXiv:1404.5244 [cs.FL], 2014.
[13] G.K. Manacher, A new linear-time "on-line" algorithm for finding the smallest initial palindrome of a string, J. ACM 22 (3) (1975) 346–351.
[14] W. Matsubara, S. Inenaga, A. Ishino, A. Shinohara, T. Nakamura, K. Hashimoto, Efficient algorithms to compute compressed longest common substrings and compressed palindromes, Theor. Comput. Sci. 410 (8–10) (2009) 900–913.
[15] M.D. McIlroy, The number of 1's in binary integers: bounds and extremal properties, SIAM J. Comput. 3 (4) (1974) 255–261.