Computing Primitively-Rooted Squares and Runs in Partial Words*

F. Blanchet-Sadri[1]　J. Lazarow[2]　J. Nikkel[3]
J. D. Quigley[4]　X. Zhang[5]

August 19, 2016

Abstract

This paper deals with two types of repetitions in strings: *squares*, which consist of two adjacent occurrences of substrings, and *runs*, which are periodic substrings that cannot be extended further to the left or right while maintaining the period. We show how to compute all the primitively-rooted squares in a given partial word, which is a sequence that may have undefined positions, called holes or wildcards, that match any letter of the alphabet over which the sequence is defined. We also describe an algorithm for computing all primitively-rooted runs in a given partial word and extend previous analyses on the number of runs to partial words.

Keywords: String algorithms; Combinatorics on words; Partial words; Primitive partial words; Primitively-rooted squares; Primitively-rooted runs.

1 Introduction

Repetitions in strings, or words, have been extensively studied, both from the algorithmic point of view and the combinatorial point of view (see, for example, [14]). Applications can be found in many important areas such as computational biology, text compression, natural language processing, to name a few [12, 30]. Repetitions are characterized by their periods, lengths, and starting positions. There are many equivalent characterizations

*This is a journal version of [10].

[1]Department of Computer Science, University of North Carolina, P.O. Box 26170, Greensboro, NC 27402–6170, USA, `blanchet@uncg.edu`

[2]Department of Mathematics, University of Texas at Austin, 1 University Station C1200 Austin, TX 78712–0233, USA, `jlazarow@utexas.edu`

[3]Department of Mathematics, Vanderbilt University, 1326 Stevenson Center, Nashville, TN 37240, USA, `jordan.k.nikkel@vanderbilt.edu`

[4]Department of Mathematics, University of Illinois - Urbana-Champaign, Altgeld Hall, 1409 W. Green Street, Urbana, IL 61801, USA, `jdquigl2@illinois.edu`

[5]Department of Mathematics, Princeton University, Fine Hall, Washington Road, Princeton, NJ 08544, USA, `xufanz@princeton.edu`

of repetitions, and in this paper a repetition in a word w is a triple (f, l, p), where $w[f..l]$ is p-periodic and the *exponent* of the repetition, $\frac{l-f+1}{p}$, is at least 2.

Squares are the special case of repetitions when $\frac{l-f+1}{p}$ is 2. In other words, a square in a word is a factor uu for some word u, called the *root* of the square. It is *primitively-rooted* or *PR* if u is *primitive*, i.e., u is not a power of another word. There can be as many as $\Theta(n \log n)$ occurrences of *primitively-rooted* squares in a word of length n, and several $O(n \log n)$ time algorithms have been developed for finding all the repetitions [2, 11, 31]. A major breakthrough was to compute them in $O(n)$ time; this was achieved in two steps: (1) all repetitions are encoded in *maximal repetitions* or *runs* and (2) there is a linear bound on the number of runs [27, 28].

A repetition (f, l, p) is maximal, or is a run, if it is non-extendible, i.e., neither $(f - 1, l, p)$ nor $(f, l + 1, p)$ are repetitions in the word w. Note that since every run has exponent at least 2, the square at the beginning of the run uniquely defines the run. This is because that square contains the starting position and period of the run, and the property of the run being maximal gives a unique ending position. A *PR-run* in w is a maximal repetition (f, l, p) with a primitive root of length p. If $w = 0001101011010110101 0$, then $w[2..18] = (01101)^{\frac{17}{5}}$ is a PR-run with period 5, root 01101, and exponent $\frac{17}{5}$ (indexing in the word starts at 0). The maximum number of runs in a string of length n is bounded from above by cn, for some constant c. A first proof was given by Kolpakov and Kucherov [27, 28] who provided an $O(n)$ time algorithm for detecting all runs. However, no constant c can be deduced from their proof. Kolpakov and Kucherov's "runs conjecture" that the maximum number of runs in a string of length n is less than n has generated bounds that have been improved over the years (upper bounds [35, 34, 13, 22, 23, 15] and lower bounds [19, 20, 21, 33, 36]). Recently, Bannai et al. [5, 4] studied runs through combinatorics of Lyndon words and finally settled the "runs" conjecture (another simple proof appears in [16]). The number of runs being less than n has applications to the analysis of any optimal algorithm for computing all repetitions.

Partial words, also referred to as *strings with don't-cares* which allow for incomplete or corrupted data [1, 18], are sequences that may contain undefined positions, called holes and represented by $\diamond$'s, that are *compatible* with, or *match*, any letter in the alphabet. *Total words* are partial words without holes. Here, a *factor* is a consecutive sequence of symbols in a partial word w, while a *subword* is a total word compatible with a factor in w. A factor uv is a square if some *completion*, i.e., a filling in of the holes with letters from the alphabet, turns uv into a total word that is a square; equivalently, u and v are compatible.

Repetitions in partial words have also recently been studied, both from the algorithmic point of view and the combinatorial point of view (see, for example, [7, 8, 9, 17, 24, 32]). However, no work has been dedicated to computing all occurrences of PR-squares and PR-runs in partial words.

The known algorithms for detecting them in total words do not extend easily to partial words, one of the most important culprits being that the compatibility relation is not transitive, as opposed to the equality relation being transitive, e.g., 0 is compatible with $\diamond$ and $\diamond$ is compatible with 1, but 0 is not compatible with 1.

So we adopt an approach, for our algorithms, based on the *large divisors* of the length n of the input partial word, i.e., divisors of n, distinct from n, whose multiples are either n itself or not divisors of n. Every distinct prime divisor i of n gives rise to exactly one large divisor of n, namely $\frac{n}{i}$, and hence the number of large divisors of n, $\omega(n)$, is the number of distinct prime divisors of n, e.g., 30 has large divisors 6, 10, and 15, giving $\omega(30) = 3$. Recently, a formula for the number of primitive partial words was given in terms of the large divisors [7]. In fact, the maximum number of holes in a primitive partial word of length n over a k-letter alphabet is $n - \omega(n) - 1$, for all $n, k \geq 2$, and this bound is tight.

The contents of our paper are as follows: In Section 2, we review a few basic concepts on partial words such as periodicity and primitivity. In Section 3, we present efficient algorithms for computing all occurrences of PR-square factors and PR-runs in any partial word over a k-letter alphabet. In Section 4, we describe an efficient algorithm for counting the number of occurrences of PR-square subwords. In Section 5, we extend previous analyses on PR-run occurrences in total words to the case of PR-run occurrences in partial words. In particular, we show a linear upper bound on the number of runs for partial words with a constant number of holes. We also show how to recursively count the exact number of microruns, those of small period, for partial words. Finally in Section 6, we conclude with some open problems.

2 Preliminary Definitions and Results

A *partial* word over an alphabet A is a sequence over the extended alphabet $A_\diamond = A \cup \{\diamond\}$, where the symbol $\diamond$ represents an undefined position or a hole. A *total* word is a partial word with no holes. For example, $0\diamond1\diamond001$ is a partial word with two holes over the alphabet $\{0, 1\}$ and 0011001 is a total word over $\{0, 1\}$. The set of positions of holes of a partial word w will be denoted by $H(w)$. A *letter* will always refer to an element of A, while a *symbol* will refer to an element of $A_\diamond$. The symbol at position i of a partial word w is denoted $w[i]$, with position numbers starting at 0. We denote by $|w|$ the *length* of w or the number of symbols in w. Two partial words u and v of equal length are *compatible*, denoted $u \uparrow v$, if $u[i] = v[i]$ for every $u[i], v[i] \in A$, and u is *contained* in v, denoted $u \subset v$, if $u[i] = v[i]$ for every $u[i] \in A$. An *upper bound* of two compatible partial words u and v is a partial word w such that $u \subset w$ and $v \subset w$. The *least upper bound* is the upper bound, denoted by $u \vee v$, such that if w is an upper bound then $(u \vee v) \subset w$. The partial words $0\diamond1\diamond0$ and $\diamond\diamond100$ are compatible and their least upper bound is $0\diamond100$, but the partial words $0\diamond1\diamond0$ and $\diamond\diamond000$ are not compatible. A *completion* of a partial word u is any total word v such that $u \subset v$.

For a positive integer p, a length-n partial word w has a *strong period* of p or is *strongly p-periodic* if $i \equiv j \bmod p$ implies $w[i] \uparrow w[j]$ for every $0 \leq i < j < n$. It has a *weak period* of p or is *weakly p-periodic* if $w[i] \uparrow w[i+p]$ for every $0 \leq i < n-p$. For example, 2 is a strong period of $\underline{0}1\underline{\diamond}1\underline{0}1$ while 2 is a weak period of $\underline{0}1\underline{\diamond}1\underline{1}1$ (the underlined positions are the ones that are congruent to 0 modulo 2). For total words, since weak periodicity implies strong periodicity, we often do not write "strong(ly)" or "weak(ly)". A partial word w is *primitive* if there exists no total word v such that $w \subset v^m$ with $m \geq 2$, equivalently, if there is no proper divisor p of $|w|$ such that w is strongly p-periodic, e.g., the partial word $01\diamond101$ is not primitive while $01\diamond111$ is primitive. Clearly, if w is primitive and $w \subset v$, then v is primitive. The next proposition states how many primitive completions a partial word with h holes over a k-letter alphabet can contain. This number is bounded by k^h, the number of completions that the word has, and any primitive partial word meets this bound.

Proposition 2.1. *Let w be a partial word with $h > 0$ holes over an alphabet of size $k \geq 2$. Then there are at least $(k-1)k^{h-1}$ primitive completions of w.*

Proof. Let w be any partial word with $h > 0$ holes over a k-letter alphabet A such that $k \geq 2$. Choose any position i such that i is a hole in w. Let $a \in A$, then consider $\hat{w}_{i,a}$ to be any completion of w with the requirement that $\hat{w}_{i,a}[i] = a$, noting that there are k^{h-1} such completions. For each letter $b \in A$, different from a, consider the completion of w that agrees with $\hat{w}_{i,a}$ except that the letter at position i is b instead of a. By [6, Corollary 6.9], at most one of these completions is non-primitive, and hence for each of the k^{h-1} completions $\hat{w}_{i,a}$, we have at least $k-1$ unique primitive completions of w. $\square$

A *factor* of a partial word w is a consecutive sequence of symbols in w, while a *subword* of w is a total word compatible with a factor in w. For example, $\diamond0\diamond$ is a factor of $00\diamond0\diamond1$ and $000, 001, 100, 101$ are the subwords of $00\diamond0\diamond1$ compatible with the factor $\diamond0\diamond$. We denote by $w[i..j)$ the factor of w starting at position i and ending at position $j-1$, and by $w[i..j]$ the factor of w starting at position i and ending at position j. A *square factor* of w is a factor of the form uv with u and v compatible. The *root* of uv is $u \vee v$. Now uv is a *PR-square factor* if its root is primitive, while uu is a *PR-square subword* if it is a total square word uu, with primitive root u, that is compatible to a square factor (not necessarily a PR-square factor). For example, if $w = 0\diamond\diamond10\diamond\diamond111$ then $0\diamond\diamond10\diamond\diamond1$ is a factor occurring at position 0 of w but it is not a PR-square factor, however, 01110111 is a PR-square subword occurring at position 0 of w. Note that $w[3..6]$ has three PR-square factors, $0\diamond$, $\diamond\diamond$, and $10\diamond\diamond$, with primitive roots 0, $\diamond$, and 10, respectively. It has four PR-square subword occurrences, 0^2, 1^2, and $(10)^2$, with 0^2 occurring twice, at positions 1 and 2.

To extend the definition of repetition, we use *strong* periodicity. A *repetition* in a partial word w is encoded by a triple (f, l, p), where $w[f..l]$ is strongly p-periodic and the *exponent* of the repetition, $\frac{l-f+1}{p}$, is at least 2. We call the integer f the first position of the repetition, the integer l the last position of the repetition, and the integer p the period of the repetition. The *root* of a repetition (f, l, p) in w is the length-p partial word u such that for all $0 \leq i < p$,

$$u[i] = w[f+i] \vee w[f+p+i] \vee \cdots \vee w[f+cp+i]$$

with the smallest integer c satisfying $f+(c+1)p+i > l$. A *maximal repetition* or a *run* is a repetition (f, l, p) such that neither $(f-1, l, p)$ nor $(f, l+1, p)$ are repetitions, that is, we cannot extend the run (with respect to period) to the left nor right. A *PR-run* is then defined as a run with primitive root. For example, in $110\diamond0\diamond0\diamond1$, $(2, 5, 2)$ is a square with root $0\diamond$, $(1, 7, 2)$ is a maximal repetition with root 10, and $(2, 7, 1)$ is a PR-run with root 0. The next proposition gives a condition for a maximal repetition to be PR.

Proposition 2.2. *A maximal repetition (f, l, p) in a partial word w is not PR if and only if there exists a maximal repetition (f, l, p') in w such that p' is a large divisor of p.*

Proof. Let w be given, and suppose that (f, l, p) is a maximal repetition in w with root u such that u is not primitive. Then there exists a word v and integer $m \geq 2$ such that $u \subset v^m$ and v can be chosen so that $|v|$ is a large divisor of $|u| = p$. Indeed, $|u| = m|v|$. If m is prime, then $|v|$ is a large divisor of $|u|$. If m is not prime, then $m = p_1 \cdots p_r$ where $p_1, \ldots, p_r$ are primes, in which case $u \subset (v^{p_1 \cdots p_{r-1}})^{p_r}$ and $|v^{p_1 \cdots p_{r-1}}|$ is a large divisor of $|u|$. But $w[f] \cdots w[l] \subset u^{\frac{l-f+1}{p}}$, so $w[f] \cdots w[l] \subset (v^m)^{\frac{l-f+1}{p}}$. Now (f, l, p) being maximal implies that $w[f-1] \not\uparrow u[p-1]$, and since $u[p-1] \subset v[|v|-1]$, $w[f-1] \not\uparrow v[|v|-1]$. Similarly, $w[l+1] \not\uparrow u[0]$, and since $u[0] \subset v[0]$, we have $w[l+1] \not\uparrow v[0]$. This implies that $(f, l, |v|)$ is a maximal repetition in w.

Similarly, suppose that (f, l, p) is a maximal repetition in w with root u and there exists a maximal repetition (f, l, p') in w with root v such that p' is a large divisor of p. Let $m = \frac{p}{p'}$, an integer by assumption. Then by definition of root, $w[f] \cdots w[l] \subset v^{\frac{l-f+1}{p'}} = (v^m)^{\frac{l-f+1}{p}}$. Then v^m is an upper bound for the root of (f, l, p) by definition of the root of a repetition, and hence $u \subset v^m$. Thus (f, l, p) does not have a primitive root. $\square$

3 Computing all PR-Square Factor and Run Occurrences

We develop efficient algorithms for computing the PR-square factor and run occurrences. In our pseudocode, we use `poll`() (respectively, `peek`()) to retrieve and remove (respectively, retrieve but not remove) the head of a first-in first-out queue.

Algorithm 3.1 Finding the positions of every PR-square factor occurrence in a partial word w of length $n \geq 2$ over a k-letter alphabet

```
Require: $SqQ$, $RepQ$ are empty queues which hold integer pairs, $isPR$ is
    a boolean array of size $n \times \lfloor n/2 \rfloor$ where each entry is initialized to $true$
1: for $m = 1, \ldots, \lfloor n/2 \rfloor$ do
2:    Procedures 3.2, 3.3, and 3.4
```

Algorithm 3.1 finds the positions of every PR-square factor occurrence in a partial word of length $n \geq 2$ over a k-letter alphabet. Let SqQ, $RepQ$ be empty queues which hold integer pairs, and $isPR$ be a boolean array of size $n \times \lfloor \frac{n}{2} \rfloor$, where each entry is initialized to *true*. The idea behind Algorithm 3.1 is to find all squares by increasing root length, and use the information from the squares of smaller root lengths to determine the primitivity of the roots of squares of larger root lengths.

Procedure 3.2 `PSFBlocks`(m)

```
Ensure: blocks and positions of PR-square factors of root length $m$
1: $start \leftarrow 0$
2: for $i = 0, \ldots, n - m - 1$ do
3:    if $w[i]$ is incompatible with $w[i + m]$ then
4:       if $i - start \geq m$ then
5:          push $(start, i + m)$ onto $SqQ$
6:          for $j = start, \ldots, i - m$ do
7:             if $isPR[j][m]$ then
8:                output that a PR-square factor of root length $m$ occurs at $j$
9:       $start \leftarrow i + 1$
10:   else if $i = n - m - 1$ and $i - start + 1 \geq m$ then
11:      push $(start, n)$ onto $SqQ$
12:      for $j = start, \ldots, i + 1 - m$ do
13:         if $isPR[j][m]$ then
14:            output that a PR-square factor of root length $m$ occurs at $j$
```

Procedure 3.2: All squares of root length m can be found easily in maximal weakly m-periodic factors of w (i.e., if we extend the factor to the left or right, it is no longer weakly m-periodic), which we refer to as *blocks*. Computing all such blocks can be done by simply iterating once through w and checking positions that are m-apart for compatibility. When an incom-

Procedure 3.3 `PSFReps`(m)

Require: *root* (respectively, *lastLetter*) is a symbol (respectively, an integer) array of length m
Ensure: maximal repetitions of root length m

```
 1: while $SqQ$ is not empty do
 2:    $block \leftarrow$ poll($SqQ$)
 3:    $start \leftarrow block.first$
 4:    $rootPos \leftarrow 0$
 5:    for $i = 0, \dots, m-1$ do
 6:       $root[i] \leftarrow w[start + i]$
 7:       if $root[i]$ is not a hole then
 8:          $lastLetter[i] \leftarrow start + i$
 9:       else
10:          $lastLetter[i] \leftarrow start$
11:    for $i = block.first + m, \dots, block.last - 1$ do
12:       if ($w[i]$ is not compatible with $root[rootPos]$) and
          ($lastLetter[rootPos] \geq start$) then
13:          push $(start, i - start)$ onto $RepQ$
14:          $start \leftarrow lastLetter[rootPos] + 1$
15:          $root[rootPos] \leftarrow w[i]$ and $lastLetter[rootPos] \leftarrow i$
16:       else if $i = block.last - 1$ then
17:          push $(start, block.last - start)$ onto $RepQ$
18:       else
19:          if $w[i]$ is not a hole then
20:             $root[rootPos] \leftarrow w[i]$ and $lastLetter[rootPos] \leftarrow i$
21:       $rootPos \leftarrow (rootPos + 1) \bmod m$
```

patibility is found, the current block is put onto a queue and the next block is started, skipping over the incompatibility. To check if a square of root length m is PR, we look at *isPR[j][m]* which indicates whether or not the square factor beginning at j with root length m is PR. The array *isPR[j][m]* is updated in *Procedure 3.4*.

Example 3.1. *To illustrate Procedure 3.2, consider the partial word*

0	1	2	3	4	5	6	7	8	9	10	11
1	0	1	$\diamond$	$\diamond$	$\diamond$	$\diamond$	0	1	$\diamond$	0	1

12	13	14	15	16	17	18	19	20	21	22
2	$\diamond$	1	2	1	1	2	0	0	0	$\diamond$

The following table lists the blocks of each root length m found by this procedure, e.g., there is one block of root length $m = 3$*, (0,18), which is a maximal weakly* 3*-periodic factor, and there are no blocks of root length* $m = 5$.

Procedure 3.4 `PSFPR`(m)

```
Ensure: mark off, using the maximal repetitions, which squares of larger
    root length are not PR
while $RepQ$ is not empty do
    $rep \leftarrow$ poll($RepQ$)
    $start \leftarrow rep.first$ and $runLength \leftarrow rep.length$
    $nextLength \leftarrow 0$
    if $RepQ$ is not empty then
        $nextRep \leftarrow$ peek($RepQ$)
        $next \leftarrow nextRep.first$ and $nextLength \leftarrow nextRep.length$
    for each prime $p$ less than or equal to $\frac{runLength}{2m}$ do
        $maxPos \leftarrow runLength - 2pm$
        if $p \leq \frac{nextLength}{2m}$ then
            $maxPos \leftarrow \min\{maxPos, next - start - 1\}$
        for $i = 0, \ldots, maxPos$ do
            $isPR[start + i][pm] \leftarrow false$
```

m					
1	$(2,7)$	$(8,10)$	$(12,14)$	$(16,17)$	$(19,22)$
2	$(0,9)$	$(13,16)$	$(19,22)$		
3	$(0,18)$				
4	$(0,10)$				
5					
6	$(0,15)$				
≥ 7					

Procedure 3.3: Finding all maximal repetitions of root length m can be done by iterating through every block, maintaining the current period, and keeping track of the last position where a letter was seen for each position in the period. When an incompatibility is found, the current repetition is put onto a queue, the period position is updated to the most recent letter, and the process continues from that same spot.

Example 3.2. *Consider again Example 3.1. Once a block is found in Procedure 3.2, Procedure 3.3 finds maximal repetitions or runs of root length m. For example for $m = 5$, it finds no maximal repetitions while for $m = 3$, it finds the maximal repetitions $(0, 11), (1, 15), (11, 18)$. To see the latter, the block $(0, 18)$, a maximal weakly 3-periodic factor, is split into three maximal repetitions, maximal strongly 3-periodic factors. In particular, the procedure keeps track of the last position where a letter was seen for the first position in the period, i.e., position 0 with letter 1. An incompatibility is found with position 12 with letter 2, so the repetition $(0, 11)$ is put onto a queue, the first position of the period is updated to letter 2, and the process continues from position 1 of the block.*

m								
1	$(2,6)$	$(3,7)$	$(8,9)$	$(9,10)$	$(12,13)$	$(13,14)$	$(16,17)$	$(19,22)$
2	$(0,9)$	$(13,16)$	$(19,22)$					
3	$(0,11)$	$(1,15)$	$(11,18)$					
4	$(0,9)$	$(3,10)$						
5								
6	$(0,11)$	$(1,15)$						
≥ 7								

Procedure 3.4: Since any square that is not PR is contained in a repetition of some root length m, using the maximal repetitions to mark off which squares of larger root lengths are not PR provides a good alternative to checking every factor individually for primitivity. For each maximal repetition found from left to right, and for each prime p, mark off the squares of root length pm contained in the current maximal repetition but not in any later maximal repetition.

Example 3.3. *Returning to Example 3.2, once a maximal repetition is found in Procedure 3.3, Procedure 3.4 updates* $isPR[j][m]$*, e.g., when considering the maximal repetition* $(2,6,1)$*,* $isPR[2][2]$ *and* $isPR[3][2]$ *are set to* $false$*. Similarly, the following are set to* $false$ *when considering the run* $(1,15,3)$*:* $isPR[1][6]$*,* $isPR[2][6]$*,* $isPR[3][6]$*, and* $isPR[4][6]$ *(note that* $isPR[5][6]$ *will be updated when considering the later repetition* $(1,15)$*).*

Theorem 3.4. *Algorithm 3.1 outputs exactly the PR-square factors.*

Proof. We claim that Procedure 3.2 finds every block of root length m. If Lines 3–9 push a block, then every position in the block has been checked for compatibility with the position m positions away, and thus the block contains squares of root length m. Moreover, the block is cut off because of an incompatibility on the right side, and it was started directly after an incompatibility on the left side, thus the block is maximal. The block is also guaranteed to have at least one square since $i - start \geq m$ implies that $i + m - start \geq 2m$, and thus the first position of the block starts a square. If a block is pushed on in Lines 10–14, then we are on the last iteration, and hence the same argument applies, noting that $i - start + 1 \geq m$ is the exact condition for when the block contains at least one square, since getting to this conditional statement means that $w[i]$ is compatible with $w[i + m]$. Furthermore, if u is a block, then the same basic logic applies and u is found by the procedure. Thus exactly every block of root length m is found by Procedure 3.2.

Next we claim that Procedure 3.3 finds all maximal repetitions of root length m. Since every such repetition is weakly m-periodic, every maximal repetition is contained in a block, so it suffices to show that the procedure finds every run within a given block. By checking compatibilities with a given root, and pushing the current repetition only when we reach the end of the block or an incompatibility with the root, we ensure that every repetition is maximal on the right. Moreover, if an incompatibility between $w[i]$ and $root[rootPos]$ is reached, then the next run must have $root[rootPos] = w[i]$, and the start of the next run must be at least one position after the previous run as well as at least one position after the last non-hole that was compatible with $root[rootPos]$. Thus the repetition cannot be extended on either side and is maximal. Moreover, since every position starts a square, every repetition found has at least length $2m$, and is thus a valid repetition. This implies that the procedure only outputs valid, maximal repetitions of root length m. Similarly it finds every such repetition.

Finally, to prove that Procedure 3.2 outputs only PR-square factors, we must prove that Procedure 3.4 updates *isPR* completely. Suppose a square uv of root length m that is not PR occurs at some position. Then $uv \subset x^{2q}$ for some x^q such that q is a proper divisor of m. If q is not a large divisor of m, then there exists a prime p such that $\frac{m}{p}$ is a multiple of q. This implies that $uv \subset (x^{\frac{q}{p}})^{2p}$, so uv is contained in a repetition that has a period length equal to a large divisor of m. Thus for every square of root length m that is not PR, there is a repetition of period length $\frac{m}{p}$ that witnesses the non-primitivity of the root of uv such that $\frac{m}{p}$ is a large divisor of m. As a result, consider a given repetition of root length m. Then for every prime p such that a factor uv of length $2pm$ occurs in the repetition, m is a large divisor of pm and is a witness to the non-primitivity of uv. Also, since two repetitions of the same period length may overlap, it suffices to only use a rightmost repetition as the witness for a given position and square root length. Therefore Procedure 3.4 updates *isPR* completely and correctly so that Procedure 3.2 outputs every PR-square factor. $\square$

Theorem 3.5. *Given a partial word w of length n, Algorithm 3.1 computes the number of occurrences of PR-square factors of w in $O(n^2 \log\log n)$ time.*

Proof. First consider Procedure 3.2. The outer loop (Line 2) iterates less than n times, and every command other than the inner loops takes constant time. After any going through the first inner loop (Line 3), *start* is always set to larger than i, and since i is always increasing, *start* takes on each value from 1 to $n - m$ at most once. Moreover, when the second inner loop (Line 10) iterates, the outer loop is on its last iteration, and hence it adds no more than $O(n)$ time. Thus Procedure 3.2 takes $O(n)$ time.

Next consider Procedure 3.3. The outer loop (Line 1) iterates over each block found by Procedure 3.2. Note that each block has at least length $2m$ since it must contain a square. The first m positions are iterated over by the first for loop (Lines 5–10), and the remaining positions are iterated over by the second for loop (Lines 11–21), both of these loops taking at most constant time per iteration. Thus it remains to show that the number of positions in all the blocks combined is $O(n)$. To see this, observe that for every block except the last, there must be m positions which cannot start squares because of at least one incompatibility. Moreover, there are two types of positions in every block: those which start a square, and those which do not but are included because they are contained in the last square of the block. The last $2m-1$ positions fit this latter category, but the first m of them cannot begin square positions. Thus at most the last $m-1$ positions are double-counted in a second block. Since the starts of every block are thus at least m-apart, there are at most $\frac{n}{m}$ blocks of root length m. Thus the number of positions double-counted is at most $\frac{n}{m}(m-1)$, which is $O(n)$. Since Algorithm 3.1 has $\lfloor \frac{n}{2} \rfloor$ iterations, the total time spent on Procedures 3.2 and 3.3 is $O(n^2)$.

Consider Procedure 3.4's total running time over every root length m. First, the primes less than n can easily be computed in $O(n^2)$ time using a sieve [3]. Second, by considering the next values from $RepQ.first$ and $RepQ.length$, we check in constant time for each prime less than or equal to the current value of $\frac{runLength}{2m}$ whether or not the positions which this repetition and the next could update have overlap, and if they do, we save the positions for the next repetition to update. Thus no two distinct repetitions of the same root length m access $isPR[i][m]$ for any position i.

Now consider how many times any given value $isPR[i][m]$ is accessed: once at the beginning when it is set to the default value $true$, once if a square of root length m occurs at position i, and once for each distinct prime factor p of m for which there is a repetition of root length $\frac{m}{p}$ that contains $w[i]\cdots w[i+2m-1]$. This gives a total of $2+\omega(m)$ updates at most, where $\omega(m)$ denotes the number of distinct prime factors of m. Summing over all root lengths at position i gives at most $\sum_{j=1}^{\lfloor \frac{n-i}{2} \rfloor} \omega(j)$ updates, which by [25] is $\Theta(\frac{n-i}{2} \log\log \frac{n-i}{2})$. Summing over all positions gives $\sum_{i=0}^{n-1} c(n-i)\log\log(n-i)$ as an upper bound, and this is in turn $O(n^2 \log\log n)$. $\square$

Remark 3.6. *Algorithm 3.1 has a best case running time of $O(n^2)$ because Procedures 3.2 and 3.3 are $O(n^2)$. By examining the algorithm, we see that Procedure 3.4 is $O(n^2)$ when the number of maximal repetitions is constant, so in this case, Algorithm 3.1 has running time $O(n^2)$.*

Algorithm 3.1 can be modified for computing all PR-runs in a partial word. The only modifications that are necessary are to remove the outputs from Procedure 3.2, to remove Procedure 3.4, and to check which maximal repetitions are PR using Proposition 2.2. Thus, all PR-runs can be found by sorting the repetitions based on their first and last positions using a bucket sort, and then checking which periods are large divisors of which for each starting and ending position. The following example illustrates this PR-runs algorithm.

Example 3.7. *Looking at Examples 3.1 and 3.2 again, we have the same maximal repetitions from Procedures 3.2 and 3.3. For $m=1$, every maximal repetition will be marked as PR. Since $(0,9,1)$ is not a PR-run, $(0,9,2)$ is marked as PR. Then $(0,9,4)$ is marked as not PR because 2 is a large divisor of 4.*

The time complexity of our algorithm is stated in the following theorem.

Theorem 3.8. *Given a partial word w of length n, all PR-run occurrences in w can be computed in $O(n^2 \log\log n)$ time.*

Proof. By the previous theorems, Procedures 3.2 and 3.3 correctly identify all maximal repetitions for all root lengths $1 \leq m \leq \lfloor \frac{n}{2} \rfloor$ in $O(n^2)$ time. Maximal repetitions are characterized by their starting position f, ending position l, and root length m. Proposition 2.2 states that a maximal repetition (f, l, p) in w is not PR if and only if there exists a maximal repetition (f, l, p') in w such that p' is a large divisor of p. We can sort the runs by their start and end positions in $O(n^2)$ time using a bucket sort. Since we process m in increasing order, the runs in each bucket are naturally ordered by increasing root length, and we examine them again in this order. To ensure that only PR-runs are counted, we check each run (f, l, p) against the already examined runs (f, l, p') with the same starting and ending positions. Since $p > p'$, we have the cases: (1) if there is no (f, l, p') such that p' is a large divisor of p, then (f, l, p) is PR; (2) if there is (f, l, p') such that p' is a large divisor of p, then (f, l, p) is not PR. For each p, we only need to check all its large divisors p', the number of which is $\omega(p)$. So it suffices to bound $\sum_{p=1}^{n} \omega(p)$, this sum is $O(n \log\log n)$. There are $O(n)$ repetitions for any fixed p. □

4 Counting all PR-Square Subword Occurrences

We now develop an efficient algorithm to count the number of PR-square subword occurrences in a given partial word.

Algorithm 4.1 Finding the number of PR-square subwords compatible with each square factor of a partial word w of length $n \geq 2$ over a k-letter alphabet

```
Require: SqQ, RepQ are empty queues, isPR is a boolean array of size
    $n \times \lfloor n/2 \rfloor$ where each entry is initialized to true, and wRoot is an array
    of symbols of length $n$
1: for $m = 1, \ldots, \lfloor n/2 \rfloor$ do
2:     Procedures 4.2, 3.3, and 3.4
```

Algorithm 4.1 finds the number of PR-square subword occurrences compatible with each square factor of a partial word w of length $n \geq 2$ over a k-letter alphabet. In addition to *SqQ*, *RepQ*, and *isPR* from Algorithm 3.1, Algorithm 4.1 requires an array *wRoot* of symbols of length n. Finding all PR-square subword occurrences reduces to first finding all square factor occurrences and determining if they are primitively-rooted, which Algorithm 3.1 does, and then finding all primitive completions of the roots of these square factor occurrences, which Procedure 4.3 does. Procedure 4.2 calls Procedure 4.3. We first outline Procedure 4.3, which takes as input a

Procedure 4.2 `PSSBlocks`(m)

```
Ensure: blocks and number of PR-square subword occurrences of root length $m$
 1: $start \leftarrow 0$
 2: for $i = 0, \dots, n-1$ do
 3:   if $i + m < n$ and $w[i] \uparrow w[i+m]$ then
 4:     $wRoot[i] \leftarrow w[i] \vee w[i+m]$
 5:   else
 6:     $wRoot[i] \leftarrow w[i]$
 7: for $i = 0, \dots, n-m-1$ do
 8:   if $w[i]$ is incompatible with $w[i+m]$ then
 9:     if $i - start \geq m$ then
10:       push $(start, i+m)$ onto $SqQ$
11:       for $j = start, \dots, i-m$ do
12:         output PC($wRoot[j] \cdots wRoot[j+m-1], m, h, k, isPR[j][m]$)
13:         if $wRoot[j]$ is a hole and $wRoot[j+m]$ is not a hole then
14:           $h \leftarrow h - 1$
15:         else if $wRoot[j]$ is not a hole and $wRoot[j+m]$ is a hole then
16:           $h \leftarrow h + 1$
17:     $start \leftarrow i + 1$
18:     $h \leftarrow 0$
19:   else if $i = n-m-1$ and $i - start + 1 \geq m$ then
20:     push $(start, n)$ onto $SqQ$
21:     for $j = start, \dots, i+1-m$ do
22:       output PC($wRoot[j] \cdots wRoot[j+m-1], m, h, k, isPR[j][m]$)
23:       if $wRoot[j]$ is a hole and $wRoot[j+m]$ is not a hole then
24:         $h \leftarrow h - 1$
25:       else if $wRoot[j]$ is not a hole and $wRoot[j+m]$ is a hole then
26:         $h \leftarrow h + 1$
27:   if $i - start < m$ and $wRoot[i]$ is a hole then
28:     $h \leftarrow h + 1$
```

partial word u of length n with h holes over a k-letter alphabet and outputs the number of primitive completions of u.

If u is primitive, then Line 2 returns the correct value and if u is non-primitive, then Line 19 does so. To see the latter, first recall that the number of primitive total words of length n over a k-letter alphabet is $\sum_{d|n} k^d \mu(\frac{n}{d})$, where μ is the Möbius function [29]. Recall that $\mu(n) = 1$ if n is a positive integer that has an even number of prime divisors but that has no squared prime divisor, $\mu(n) = -1$ if n has an odd number of prime divisors but has no squared prime divisor, and $\mu(n) = 0$ if n has a squared prime divisor. Since Procedure 4.3 is called by Algorithm 4.1, which takes at least $\Omega(n^2)$ time, we can take $O(n^2)$ time to pre-compute all divisors of the integers 1 to n as well as their values for the Möbius function.

Procedure 4.3 PC($u, n, h, k, isPrimitive$)

```
Require: $u$ is a partial word of length $n$ with $h$ holes over a $k$-letter alphabet, $isPrimitive$ is true if $u$ is primitive
Ensure: the number of primitive completions of $u$
1: if $isPrimitive$ then
2:     return $k^h$
3: else
4:     $result \leftarrow 0$
5:     for each divisor $d$ of $n$ in increasing order do
6:         $isCompatible \leftarrow true$
7:         $i \leftarrow 0$
8:         let $period$ be a symbol array of length $d$
9:         while $i < n$ and $isCompatible$ do
10:            if $i < d$ then
11:                $period[i] \leftarrow u[i]$
12:            else
13:                if $period[i \bmod d]$ is a hole and $u[i]$ is not a hole then
14:                    $period[i \bmod d] \leftarrow u[i]$
15:                $isCompatible \leftarrow period[i \bmod d] \uparrow u[i]$
16:            $i \leftarrow i + 1$
17:        if $isCompatible$ then
18:            $result \leftarrow result + k^{\#\text{ holes in period}}\mu(\frac{n}{d})$
19:    return $result$
```

Suppose that u is non-primitive. Then u is strongly d-periodic for some proper divisor d of n. Thus we can classify every primitive completion of u by the partial word v with the maximum number of holes such that $u \subset v^{\frac{n}{d}}$. Lines 5–18 consider periods v whose lengths are divisors of n and Line 18 counts the primitive completions of u corresponding to each period found. Note that it is possible to double-count primitive completions of u (e.g., $11\diamond\diamond1\diamond$). So, it is a matter of using inclusion-exclusion to ensure no double-counting. We can apply the number-theoretic inclusion-exclusion (involving the Möbius function). To find the number of primitive completions of u it suffices to find, for each divisor d of n, the number of total words of length n over a k-letter alphabet that are an $\frac{n}{d}$-th power, not necessarily primitively-rooted, compatible with u, and then sum up these results with the appropriate $\mu(\frac{n}{d})$ coefficients. Assuming that all the divisors of n have been pre-computed as well as their values for the Möbius function, this requires stepping through u once for each divisor which takes $O(nd(n))$ time, where $d(n)$ is the number of divisors of n. Procedure 4.3 is run $O(n)$ times for any fixed $|u|$.

Now, consider Procedure 4.3 when applied in Line 12 of Procedure 4.2, for two consecutive values of j. Then the body of the loop of Procedure 4.3 actually does almost the same thing for both calls. This redundancy can be easily replaced with a sliding window approach, which already proved useful in Procedure 3.2. For any factor of length M and any divisor d of M, we count the number of total words which are an $\frac{M}{d}$-th power, not necessarily primitively-rooted, and are compatible with the factor. Since we are ultimately interested in PR-squares, we can restrict to even values of M and divisors d of $\frac{M}{2}$. These values appear in the number-theoretic inclusion-exclusion, with coefficients $\mu(\frac{M}{2d})$, counting the PR-square subwords compatible with a given factor.

For fixed M and d, we can compute these values for all length-M factors in $O(n)$ time. It suffices to have a sliding window of fixed length M, which for any remainder modulo d tracks the situation at positions inside the sliding window with that remainder modulo d. There can be either a conflict (at least two different letters), just one letter, or just holes. To maintain the situation for a fixed remainder, it suffices to store the last two letters encountered along with the positions where they were encountered for the last time. Then a conflict in some remainder is equivalent to $isCompatible = false$ in Procedure 4.3, and if there is no conflict, the answer is $k^{\#\text{ of remainders with just holes}}$. The sliding window needs to be applied $\sum_{M=2}^{n} d(M)$ times.

This sum is $O(n \log n)$; there are $O(\frac{n}{i})$ multiplicities of i not exceeding n, this leads to the harmonic series and $H_n = \sum_{i=1}^{n} \frac{1}{i} = O(\log n)$.

Theorem 4.1. *Given a partial word w of length n, Algorithm 4.1 computes the number of occurrences of PR-square subwords of w in $O(n^2 \log n)$ time.*

Example 4.2. *Reexamining* $101\diamond\diamond\diamond\diamond01\diamond012\diamond12112000\diamond$ *from Examples 3.1, 3.2 and 3.3, the square occurrence* $1\diamond\diamond\diamond$ *at position 2 has a PR-square completion* 1010*, but was ruled out before because it is not a PR-square factor. Similarly,* $\diamond\diamond\diamond\diamond$ *at position 3 has PR-square completions* 0101 *and* 1010*, but was ruled out before because it is not a PR-square factor. Additionally,* $01\diamond\diamond\diamond\diamond01\diamond012$ *at position 1 is not a PR-square factor, but it has PR-square completions* 011012011012 *and* 010012010012*.*

We end this section with the following remarks.

Remark 4.3. *One might try to apply dynamic programming to find the runs in all the algorithms above, but since it must be at least quadratic, it will not improve the time complexity of the current algorithms. Also, to keep the transition in dynamic programming efficient, extra work has to be done, which makes it essentially the same as the current algorithms.*

Remark 4.4. *If h is large, the result does not fit a RAM word, but instead we can represent it as $\sum_{i=0}^{h} a_i k^i$ for some (small) integer coefficients a_i.*

5 Counting PR-Run Occurrences in Partial Words

The proof from [4], that the number of maximal repetitions or runs in total words of length n is bounded from above by n, is based on combinatorics of

Lyndon words. It is not at all clear how the concept of Lyndon words can be extended to partial words without losing the interesting combinatorial properties that Lyndon words have. As to the proof from [16], the key observation is that each run is associated with its maximal suffix according to one of two lexicographic orderings, but in the case of partial words it is not clear how to define these orderings without losing properties such as the border-freeness of the prefix of the associated maximal suffix that has length the period of the run. So we extend the earlier analyses of runs in total words from [22, 23], which provide insights into the structure of runs in partial words.

Denote by $\mathtt{r}_{\leq p}(w)$ (respectively, $\mathtt{r}_{\geq p}(w)$) the number of occurrences of PR-runs with period at most (respectively, at least) p in a partial word w; we call them *microruns* (respectively, *macroruns*). For brevity, we write $\mathtt{r}(w)$ to indicate the number of occurrences of PR-runs in w, i.e., $\mathtt{r}(w) = \mathtt{r}_{\leq \lfloor |w|/2 \rfloor}(w)$. For example, the partial word $w = 0101\diamond0100$ has three runs of period 1, $w[3..4] = 1\diamond$, $w[4..5] = \diamond0$ and $w[7..8] = 00$, two runs of period 2, $w[0..4] = 0101\diamond$ and $w[4..7] = \diamond010$, and one run of period 3, $w[2..8] = 01\diamond0100$, so $\mathtt{r}_{\leq 1}(w) = 3$, $\mathtt{r}_{\leq 2}(w) = 5$, and $\mathtt{r}_{\leq 3}(w) = 6 = \mathtt{r}(w)$. Given integers $h \geq 0$ and $n \geq 2$, let

$$\mathtt{runs}_{\leq p}(h,n,k) = \max\{\mathtt{r}_{\leq p}(w) \mid |H(w)| = h, |w| = n\},$$

$$\mathtt{runs}_{\geq p}(h,n,k) = \max\{\mathtt{r}_{\geq p}(w) \mid |H(w)| = h, |w| = n\},$$

where the maximum is taken over all partial words w of length n with h holes over an alphabet of fixed size k. Again for brevity, we let

$$\mathtt{runs}(h,n,k) = \max\{\mathtt{r}(w) \mid |H(w)| = h, |w| = n\},$$

where the maximum is taken over all partial words over an alphabet of fixed size k. Thus, the maximum number of PR-run occurrences over all total words of length n over an alphabet of fixed size k is denoted $\mathtt{runs}(0,n,k)$, which agrees with $\mathtt{runs}_{\leq \lfloor n/2 \rfloor}(0,n,k)$. Similarly,

$$\mathtt{runs}(h,n,k) = \mathtt{runs}_{\leq \lfloor n/2 \rfloor}(h,n,k).$$

When $k = 2$, we often drop the "k" from the above notations as well as later notations related to runs.

The following theorem gives an upper bound for $\mathtt{runs}(h,n,k)$ when h and k are constants.

Theorem 5.1. *For fixed $k \geq 2$ and fixed $h \geq 0$, $\mathtt{runs}(h,n,k) = O(n)$.*

Proof. Let w be a partial word of length n with h holes over a k-letter alphabet. If $h = 0$, then we have from [28] that $\mathtt{r}(w)$ is $O(n)$. Otherwise, index the completions of w as $\hat{w}_i$, where $1 \leq i \leq k^h$. Then $\sum_{i=1}^{k^h} \mathtt{r}(\hat{w}_i) \geq \mathtt{r}(w)$. By the pigeonhole principle, there exists some $1 \leq j \leq k^h$ such that $\mathtt{r}(\hat{w}_j) \geq \frac{1}{k^h}\mathtt{r}(w)$. From [4], we have that $\mathtt{r}(\hat{w}_j) \leq n$, which in turn shows that $\mathtt{r}(w) \leq k^h n$. Thus for fixed $h \geq 0$ and $k \geq 2$, $\mathtt{runs}(h,n,k) = O(n)$. $\square$

The following proposition holds.

Proposition 5.2. *Let $k \geq 2$ be fixed. Let u be a partial word with one hole over a k-letter alphabet A and v be a total word over A, with v non-empty. Then there exist a partial word u' with one hole over A and a total word v' over A such that $|u'| + |v'| = |u| + |v|$, $|v'| < |v|$, and $\mathtt{r}(u') + \mathtt{r}(v') \geq \mathtt{r}(u) + \mathtt{r}(v)$.*

Proof. First, we discuss some terminology used in the proof. As observed in [23], some runs of two total words u and v are *merged* in uv only when there exists a total word r such that r^2 is a suffix of u and r^2 is a prefix of v. If u is a partial word with only one hole and v is a total word, a similar statement holds: some runs of u and v are *merged* in uv only when there exists a total word r such that r^2 is compatible with a suffix of u and r^2 is a prefix of v. In this case, we are guaranteed that at least one of the two halves of the suffix of length $|r|^2$ of u is equal to r, and that half defines the root of the run. But if u has more than one hole and v is a total word, this is no longer true as can be seen with $u = 0\diamond\diamond$ and $v = 11$. The run in u and the run in v have both period 1 but cannot be merged, although a total word r satisfying the above property exists; the runs in uv are $(0, 2, 1)$ along with $(1, 4, 1)$ which is a new run that gets *created* in uv, we also say the run in v, which gets lost in uv, gets *extended* through uv.

Now, let u be a partial word with one hole over a k-letter alphabet A and v be a total word over A, with v non-empty. Suppose that $|v| = 1$. By letting $u' = uv$ and $v' = \varepsilon$, we can see that $\mathtt{r}(u') + \mathtt{r}(v') = \mathtt{r}(uv) \geq \mathtt{r}(u) + \mathtt{r}(v) = \mathtt{r}(u)$. Here a run in u *corresponds* to a run in u' in the sense that the run in u' is either that same run in u contained completely in the u-segment of u' or the run in u' is extending that run in u through the v-segment of u'. So suppose that $|v| > 1$.

Write $u = xw_\diamond$ and $v = wy$, where $w_\diamond$ is the largest suffix of u compatible with a prefix of v (denoted by w). We consider the case when the hole is in $w_\diamond$; the case when the hole is in x is simpler. Take $u' = xw_\diamond y$ and $v' = w$. We have $|u'| + |v'| = |u| + |v|$ and $|v'| \leq |v|$. We assume v is not compatible with a suffix of u; otherwise, we can rewrite v as $\varphi(v)$ using an isomorphism φ that maps A onto A and we apply the process with $\varphi(v)$ instead of v (note that since $|v| > 1$ and there is only one hole in u, there is at least one letter in $\varphi(v)$ that corresponds to a different letter in the suffix of u of length $|v|$, and so using $\varphi(v)$ instead of v guarantees $|v'| < |\varphi(v)|$, e.g., if $u = 10\diamond$ and $v = 101$ then $\varphi(v) = 010$ yields $v' = 01$, but if $u = 0\diamond\diamond$ and $v = 11$, there is no isomorphism that works). So $|v'| < |v|$.

To see that $\mathtt{r}(u') + \mathtt{r}(v') \geq \mathtt{r}(u) + \mathtt{r}(v)$, first assume that there is no PR-run in v. So there is none in v'. The inequality then follows since every run in u corresponds to one in u'.

Now every PR-run in u corresponds to a run in u' (with same starting positions). Then a PR-run in v either occurs in w and thus occurs in v', or it does not occur in w and thus corresponds to a PR-run in u' (it is possible that the run in v has a letter where the corresponding run in u' has a hole). The only case of $u' = xw_\diamond y$ and $v' = w$ that can create a problem with the inequality $\mathtt{r}(u') + \mathtt{r}(v') \geq \mathtt{r}(u) + \mathtt{r}(v)$ is when y has extended a PR-run present in $xw_\diamond$. That is, removing y from v destroys a PR-run started in w from v but the corresponding PR-run that is created in $u' = xw_\diamond y$ extends a PR-run already present in $u = xw_\diamond$. Such an example is seen when $u = 0101\diamond$ and $v = 0101$. Then $x = 01$, $w_\diamond = 01\diamond$, $w = 010$, and $y = 1$, so that $u' = 0101\diamond 1 \subset (01)^3$ and $v' = 010$. Thus $\mathtt{r}(u') + \mathtt{r}(v') = 2 < \mathtt{r}(u) + \mathtt{r}(v) = 3$, i.e., we lost a run. We claim that letting $u' = uv = xw_\diamond wy$ and $v' = \varepsilon$ solves this case, i.e., $\mathtt{r}(u') + \mathtt{r}(v') = \mathtt{r}(uv) \geq \mathtt{r}(u) + \mathtt{r}(v)$, except for a special subcase discussed below.

To show our claim, let r be the root of a PR-run which occurs in $v = wy$ but which does not occur in w, and which is such that adding y to $u = xw_\diamond$ extends a PR-run with period $|r|$ in u. Note that r is a total word. Note also that w contains r (since otherwise we would not have w as the largest prefix of v that is compatible with a suffix of u). Let $w = t_2 r s_1$ and $y = t_1 y'$ for some total words s_1, t_1, s_2, t_2, y' such that $r = s_1 t_1 = s_2 t_2$. Then $u = xw_\diamond \subset xw = xt_2 r s_1$ and $v = t_2 r r y'$. Note that this containment $\subset$, as well as the other $\subset$'s in the proof, cannot be replaced by $=$'s because u has a hole. We see that $|s_1|$ must be strictly less than $|s_2|$ unless we introduce a PR-run rooted by $t_2 s_2$.

Let us consider the case when $w = (r')^p$, where r' is a total word and $p \geq 2$ is an integer. The analysis of this special case of w being a power is needed later in the proof. Set $w = t_2 s_1 z_0 t_2 s_1$, where $r = s_1 z_0 t_2$ for some $z_0 \neq \varepsilon$. Then $t_2 s_1 = (r')^q s$, where $q \geq 0$ and $r' = st$ for some s, t. Since both r' and $t_2 s_1$ are suffixes of w, we can write $r' = st = t's$ for some t'. We can also write $z_0 = t(r')^{p-2q-2}t'$. We deduce that $t' = z'z$, $t = zz'$, and $s = (z'z)^n z'$ for some z, z' and $n \geq 0$.

Suppose towards a contradiction that $t = t'$. Then $zz' = z'z$, so set $z = f^{m_1}$ and $z' = f^{m_2}$ for some word f and non-negative integers m_1, m_2. Furthermore, $t = t' = f^{m_1+m_2}$, $t_2 = f^m f'$ and $s_1 = f'' f^{m'}$ for some words f', f'' and integers m, m' satisfying $f = f'f''$ and $m + m' + 1 = q(n(m_1 + m_2) + m_2 + m_1 + m_2) + n(m_1 + m_2) + m_2$. We obtain that $r = s_1 z_0 t_2$ is not primitive, being a power of $f''f'$, a contradiction. This implies that $z \neq z'$ and $z, z' \neq \varepsilon$. This also implies that $q = 0$ and $z_0 = t(st)^{p-2}t'$.

If we expand r out, we get $r = s_1 t (st)^{p-2} t' t_2 = s_1 zz'((z'z)^n z'zz')^{p-2} z'z t_2$. Thus,

$$\begin{aligned}
u &\subset xt_2 r s_1 \\
&\subset xt_2 s_1 zz'((z'z)^n z'zz')^{p-2} z'z\underline{t_2 s_1} \\
&\subset x(z'z)^n z'zz'((z'z)^n z'zz')^{p-2} z'z\underline{(z'z)^n z'}, \\
v &= t_2 r r y' \\
&= \underline{t_2 s_1} zz'((z'z)^n z'zz')^{p-2} z'z t_2 s_1 zz'((z'z)^n z'zz')^{p-2} z'z t_2 y' \\
&= \underline{(z'z)^n z'} zz'((z'z)^n z'zz')^{p-2} z'z(z'z)^n z'zz'((z'z)^n z'zz')^{p-2} z'z t_2 y'.
\end{aligned}$$

So in this case, we add a new PR-run in $u' = uv$, contained in the underlined segment from the end of xt_2rs_1 to the beginning of t_2rry', and having root given by $t_2s_1 = (z'z)^n z'$, forming a square, except for the special subcase when $n = 0$ and z' is a prefix or a suffix of z. In this subcase, the underlined $(z'z)^n z'(z'z)^n z' = z'z'$ is actually extending a run with root z' already present in v or in u.

Let us discuss this subcase. Say z' is a suffix of z (the subcase when $n = 0$ and z' is a prefix of z is handled similarly). Set $z = z''z'$ for some $z'' \neq \varepsilon$ (note that $z'' \neq z'$ since $t \neq t'$). Here

$$\begin{array}{rcl} u & \subset & xz'z''z'z'(z'z''z'z')^{p-2}z'z''\underline{z'z'}, \\ v & = & \underline{z'}z''z'z'(z'z''z'z')^{p-2}z'z''z'z'z''z'z'(z'z''z'z')^{p-2}z'z''z't_2y'. \end{array}$$

So the prefix z' of v extends a run with root z' already in u. Example 5.4 exhibits two partial words $u = xw_\diamond$ and $v = wy$, where $n = 0$ and z' is a suffix of z, such that neither $u' = uy$ and $v' = w$, nor $u' = uv$ and $v' = \varepsilon$ give a solution. In such examples however, we can check that there is a decomposition of w, $w = (r')^{p_1}(r')^{p_2}$ with $p_1 + p_2 = p$, such that $u' = u(r')^{p_2}y = xw_\diamond(r')^{p_2}y \subset x(r')^p(r')^{p_2}y$ and $v' = (r')^{p_1}$ solve this special subcase. Indeed, every PR-run in $xw_\diamond$ corresponds to a run in $xw_\diamond(r')^{p_2}y$ (with same starting positions), while a PR-run in $v = (r')^p y = (r')^{p_1+p_2}y$ either has two periods that occur in the prefix $(r')^{p_1}$ and is thus in correspondence with a run in v', or it does not have two periods that occur in the prefix $(r')^{p_1}$ and is thus in correspondence with a PR-run in the suffix of length $|v|$ of $xw_\diamond(r')^{p_2}y$. This completes the special case when w is a power.

Let us now show that the extension of a PR-run present in v through $u' = uv$ cannot happen without at least creating a new PR-run. Denote the root of the extended run by r'. We might as well have $(r')^2$ in w (since otherwise we would not have w as the largest prefix of v that is compatible with a suffix of u). Then $w = (r')^p$, where $p \geq 2$, and we will have added a new PR-run by the previous analysis of the case of w being a power (except for the special subcase of $n = 0$ that we discussed separately).

In any case, we still preserve the total number of PR-runs and prove the result. □

The following examples illustrate the proof of Proposition 5.2.

Example 5.3. *Consider* $u = 1001010\diamond101$ *and* $v = 1011010110$. *We have* $x = 10010$*,* $w_\diamond = 10\diamond101$*,* $w = 101101$*, and* $y = 0110$*. Setting* $u' = uy = xw_\diamond y = 1001010\diamond1010110$ *and* $v' = w = 101101$ *leads to* $\texttt{runs}(u') + \texttt{runs}(v') < \texttt{runs}(u) + \texttt{runs}(v)$*; the runs in* u' *are*

$$(1,2,1),(6,7,1),(7,8,1),(12,13,1),(2,7,2),(7,12,2),$$

$$(4,9,3),(5,10,3),(0,11,5),(3,14,5),$$

the runs in v' *are*

$$(2,3,1),(0,5,3),$$

the runs in u are

$$(1,2,1),(6,7,1),(7,8,1),(2,7,2),(7,10,2),(4,9,3),(5,10,3),(0,10,5),$$

and the runs in v are

$$(2,3,1),(7,8,1),(3,7,2),(0,5,3),(0,9,5).$$

Here, $r = 10110$ is the root of a run which occurs in $v = wy$ but which does not occur in w, and which is such that adding y to $u = xw_\diamond$ extends a PR-run with period 5 in u. We are in the case when w is a square. According to our notation, $s_1 = 1, t_1 = 0110, s_2 = 10110, t_2 = \varepsilon$, and $y' = \varepsilon$.

But setting $u' = uv = 1001010\diamond1011011010110$ and $v' = \varepsilon$ leads to $\texttt{runs}(u') + \texttt{runs}(v') \geq \texttt{runs}(u) + \texttt{runs}(v)$. Here the runs in u', depicted below,

0	1	2	3	4	5	6	7	8	9	10
1	0	0	1	0	1	0	$\diamond$	1	0	1

11	12	13	14	15	16	17	18	19	20
1	0	1	1	0	1	0	1	1	0

are

$$(1,2,1),(6,7,1),(7,8,1),(10,11,1),(13,14,1),(18,19,1),$$
$$(2,7,2),(7,10,2),(14,18,2),(4,9,3),(5,16,3),(0,10,5),(11,20,5).$$

According to our notation, $w = t_2s_1z_0t_2s_1 = 101101$ implies that $t = 01$ and $t' = 10$. So $z = 0$, $z' = 1$, and $n = 0$. Here z' is neither a prefix nor a suffix of z. So a new run is added in u', i.e., $(10,11,1)$, having root given by $t_2s_1 = z' = 1$.

Example 5.4. *Next consider the partial words $u = 100010001001000\diamond000100$ and $v = 0100010001001000100010$. Here we have $x = 1000100010$, $w_\diamond = 01000\diamond000100$, $w = 010001000100$, and $y = 1000100010$. Setting $u' = uy = xw_\diamond y$ and $v' = w$ leads to $\texttt{runs}(u') + \texttt{runs}(v') < \texttt{runs}(u) + \texttt{runs}(v)$; the runs in u' are*

$$(1,3,1),(5,7,1),(9,10,1),(12,18,1),(20,21,1),(23,25,1),$$
$$(27,29,1),(6,13,3),(17,24,3),(0,10,4),(9,21,4),$$
$$(20,31,4),(2,17,7),(13,28,7),(0,31,11),$$

the runs in v' are

$$(2,4,1),(6,8,1),(10,11,1),(0,11,4),$$

the runs in u are

$$(1,3,1),(5,7,1),(9,10,1),(12,18,1),(20,21,1),$$

$$(6,13,3),(0,10,4),(9,21,4),(2,17,7),(0,21,11),$$

and the runs in v are

$$(2,4,1),(6,8,1),(10,11,1),(13,15,1),(17,19,1),$$

$$(7,14,3),(0,11,4),(10,21,4),(3,18,7),(0,21,11).$$

Here, $r = 01000100010$ is the root of a run which occurs in $v = wy$ but which does not occur in w, and which is such that adding y to $u = xw_\diamond$ extends a PR-run with period 11 in u. We are in the case when w is a cube. According to our notation, $s_1 = 0, t_1 = 1000100010, s_2 = 01000100010, t_2 = \varepsilon$, and $y' = \varepsilon$. Here, $w = t_2s_1z_0t_2s_1 = 010001000100$ implies that $t = 100$ and $t' = 010$. So $z = 10$, $z' = 0$, $n = 0$, and z' is a suffix of z.

Setting $u' = uv$ and $v' = \varepsilon$ leads to $\texttt{runs}(u') + \texttt{runs}(v') < \texttt{runs}(u) + \texttt{runs}(v)$. Here the runs in u' are

$$(1,3,1),(5,7,1),(9,10,1),(12,18,1),(20,22,1),(24,26,1),$$

$$(28,30,1),(32,33,1),(35,37,1),(39,41,1),(6,13,3),(29,36,3),$$

$$(0,10,4),(9,33,4),(32,43,4),(2,17,7),(25,40,7),(0,21,11),(22,43,11).$$

The run $(9,21,4)$ in u and the run $(0,11,4)$ in v got merged to create the run $(9,33,4)$ in u'. No new run having root $t_2s_1 = z' = 0$ is added in u', since the run $(20,22,1)$ in u' is an extension of the run $(20,21,1)$ in u.

However, setting $u' = xw_\diamond0100y$ and $v' = 01000100$ leads to $\texttt{runs}(u') + \texttt{runs}(v') \geq \texttt{runs}(u) + \texttt{runs}(v)$. Here the runs in u', depicted below,

0	1	2	3	4	5	6	7	8	9
1	0	0	0	1	0	0	0	1	0

10	11	12	13	14	15	16	17	18	19	20	21
0	1	0	0	0	$\diamond$	0	0	0	1	0	0

22	23	24	25	26	27	28	29	30	31	32	33	34	35
0	1	0	0	1	0	0	0	1	0	0	0	1	0

are

$$(1,3,1),(5,7,1),(9,10,1),(12,18,1),(20,22,1),(24,25,1),$$

$$(27,29,1),(31,33,1),(6,13,3),(21,28,3),(0,10,4),(9,25,4),$$

$$(24,35,4),(2,17,7),(17,32,7),(0,21,11),(13,35,11),(0,35,15),$$

and the runs in v' are

$$(2,4,1),(6,7,1),(0,7,4).$$

Every PR-run in $u = xw_\diamond$ corresponds to a run in $u' = xw_\diamond r'y$ (with same starting positions), while a PR-run in $v = (r')^3y$ either has two periods that occur in the prefix $(r')^2$ and is thus in correspondence with a run in v', or *it does not have two periods that occur in the prefix $(r')^2$ and is thus in correspondence with a PR-run in the suffix of length $|v|$ of $xw_\diamond r'y$. Note that the run $(20, 21, 1)$ in u corresponds to the run $(20, 22, 1)$ in u', and the run $(9, 21, 4)$ in u corresponds to the run $(9, 25, 4)$ in u'. The run $(6, 8, 1)$ in v, which has two periods of length 1 in the prefix 01000100, is an extension of the run $(6, 7, 1)$ in v', and the run $(0, 11, 4)$ in v, which has two periods of length 4 in the prefix 01000100, extends the run $(0, 7, 4)$ in v'. The run $(0, 21, 11)$ in v corresponds to the run $(13, 35, 11)$ in u'.*

We now can prove the following.

Proposition 5.5. *Let $k \geq 2$ be fixed.*

- *For every m and n, the inequality*

$$\texttt{runs}(1, m+n, k) \geq \texttt{runs}(1, m, k) + \texttt{runs}(0, n, k)$$

  holds.

- *For every p, m, and n, the inequality*

$$\texttt{runs}_{\leq p}(1, m+n, k) \geq \texttt{runs}_{\leq p}(1, m, k) + \texttt{runs}_{\leq p}(0, n, k)$$

  holds.

Proof. We prove the first statement (the second one is proved similarly). Recursively apply Proposition 5.2, letting u_0 be a witness over a k-letter alphabet for $\texttt{runs}(1, m, k)$ and v_0 be a witness for $\texttt{runs}(0, n, k)$. We then get a sequence of pairs

$$(u_0, v_0), (u_1, v_1), \ldots, (u_i, v_i)$$

with $m+n = |u_0|+|v_0| = \cdots = |u_{i-1}|+|v_{i-1}| = |u_i|$, $n = |v_0| > |v_1| > \cdots > |v_i| = 0$, and $\texttt{r}(u_i) \geq \texttt{r}(u_{i-1}) + \texttt{r}(v_{i-1}) \geq \cdots \geq \texttt{r}(u_0) + \texttt{r}(v_0)$. Furthermore,

$$\texttt{runs}(1, m+n, k) \geq \texttt{r}(u_i) \geq \texttt{r}(u_0) + \texttt{r}(v_0) = \texttt{runs}(1, m, k) + \texttt{runs}(0, n, k).$$

□

Next, for partial words u and v, denote by $\texttt{nr}_{\leq p}(u, v)$ the number of PR-run occurrences with period at most p that are in uv but that are not in u nor in v, i.e., the *new runs* between u and v. More precisely, a PR-run of uv with period $q \leq p$ belongs to exactly one of the following two cases: (1) it has a factor that is a PR-run with two periods q that are completely contained in u, or in v, or in both; (2) it has less than two periods q in both u and v, in which case it is called a new run. Note that $\texttt{r}_{\leq p}(uv) \leq \texttt{r}_{\leq p}(u) + \texttt{r}_{\leq p}(v) + \texttt{nr}_{\leq p}(u, v)$. Given an integer $h \geq 0$, let

$$\texttt{nr}_{\leq p}(h, k) = \max\{\texttt{nr}_{\leq p}(u, v) \mid |H(uv)| = h, |u| = |v| = 2p-1\},$$

where the maximum is taken over all partial words over a k-letter alphabet. Table 1 gives examples of binary partial words that achieve the $\texttt{nr}_{\leq p}(1)$ values for small periods p.

Table 1: Some $\mathtt{nr}_{\leq p}(1,2) = \mathtt{nr}_{\leq p}(1)$ values for small periods p with binary partial words u, v such that $\mathtt{nr}_{\leq p}(u,v) = \mathtt{nr}_{\leq p}(1)$.

p	u	v	$\mathtt{nr}_{\leq p}(1)$	new runs
1	$\diamond$	1	1	(0,1,1)
2	10$\diamond$	101	2	(2,3,1), (2,5,2)
3	0110$\diamond$	10110	3	(0,9,3), (4,5,1), (4,7,2)
4	010001$\diamond$	0110010	5	(0,7,4), (3,8,3), (4,10,3), (5,12,4), (6,7,1)

Proposition 5.6. *Let $k \geq 2$ be fixed. Then for every partial words u and v over a k-letter alphabet with $|H(uv)| = 1$, we have $\mathtt{nr}_{\leq p}(u,v) \leq \mathtt{nr}_{\leq p}(1,k)$.*

Proof. Any partial words u and v over a k-letter alphabet A with $|u|+|v| > 4p-2$ can have the same new PR-run occurrences realized by partial words u' and v' over A of length $2p-1$. And similarly, any partial words u and v over a k-letter alphabet A with $|u|+|v| < 4p-2$ can have their new PR-run occurrences realized by partial words u' and v' over A of length $2p-1$. $\square$

Proposition 5.7. *Let $k \geq 2$ be fixed. Then for every p, m, and n, $m \geq n$,*

$$\mathtt{runs}_{\leq p}(1, m+n, k) \leq \mathtt{runs}_{\leq p}(1,m,k) + \mathtt{runs}_{\leq p}(0,n,k) + \mathtt{nr}_{\leq p}(1,k).$$

Proof. Let u be a partial word over a k-letter alphabet A with one hole of length m and v be a total word over A of length n such that $\mathtt{r}_{\leq p}(uv) = \mathtt{runs}_{\leq p}(1, m+n, k)$. Then $|H(uv)| = 1$ and

$$\begin{aligned}\mathtt{runs}_{\leq p}(1,m+n,k) &= \mathtt{r}_{\leq p}(uv)\\ &\leq \mathtt{r}_{\leq p}(u) + \mathtt{r}_{\leq p}(v) + \mathtt{nr}_{\leq p}(u,v)\\ &\leq \mathtt{runs}_{\leq p}(1,m,k) + \mathtt{runs}_{\leq p}(0,n,k) + \mathtt{nr}_{\leq p}(1,k)\end{aligned}$$

by Proposition 5.6. $\square$

Note that the proof of Proposition 5.5 is not valid for the case $m < n$: if the value $\mathtt{runs}(1, m+n, k)$ is achieved with a partial word of length $m+n$ which contains a hole at a position in the range $[m..n-1]$, then for any partition of this partial word in factors of lengths m and n the hole is contained in the factor of length n.

Finally, we count the exact number of microruns. We do so by considering $\mathtt{nr}_{\leq p}(u,a)$, where u is a partial word over a k-letter alphabet A and $a \in A_\diamond$. A PR-run of ua with period $q \leq p$ belongs to exactly one of the following two cases: (1) it has at least two periods q in u; (2) it has less than two periods q in u, in which case it is a new run between u and a. Note that $\mathtt{r}_{\leq p}(ua) = \mathtt{r}_{\leq p}(u) + \mathtt{nr}_{\leq p}(u,a)$. Also note that $\mathtt{nr}_{\leq p}(u,a) = \mathtt{nr}_{\leq p}(u[|u|-2p..|u|), a)$ since any new run between u and a

with period $q \leq p$ has at most $2q-1 \leq 2p-1$ symbols in u and any run of u extended by a with period $q \leq p$ has at least $2q \leq 2p$ symbols in the suffix of u of length $2p$.

Let $h \geq 0$ and $n \geq 2p$ be integers and let v be a partial word with at most h holes of length $2p$ over a k-letter alphabet. Define

$$f_{p,n,h,k}(v) = \max\{\mathtt{r}_{\leq p}(uv) \mid |H(uv)| = h, |uv| = n\}, \tag{1}$$

where the maximum is taken over all partial words over that k-letter alphabet. Using the following proposition, we can recursively compute

$$\mathtt{runs}_{\leq p}(h,n,k) = \max_{|v|=2p} f_{p,n,h,k}(v).$$

Proposition 5.8. *Let $k \geq 2$ be fixed. Let w be a partial word of length $2p-1$ over a k-letter alphabet A, let $a \in A_\diamond$, and let $n \geq 2p$ be an integer. Suppose that wa has at most h holes. Then*

$$f_{p,n+1,h,k}(wa) = \begin{cases} \max_{b \in A_\diamond}\{f_{p,n,h,k}(bw) + \mathtt{nr}_{\leq p}(bw,a)\} & \textit{if } a \in A; \\ \max_{b \in A_\diamond}\{f_{p,n,h-1,k}(bw) + \mathtt{nr}_{\leq p}(bw,a)\} & \textit{if } a = \diamond. \end{cases}$$

Proof. Let us first assume that $a \in A$. We use Eq. (1) to obtain

$$f_{p,n+1,h,k}(wa) = \max\{\mathtt{r}_{\leq p}(uwa) \mid |H(uwa)| = h, |uwa| = n+1\}.$$

Rewriting u as vb, where $b \in A_\diamond$, note that $|bw| = 2p$ and $\mathtt{nr}_{\leq p}(vbw,a) = \mathtt{nr}_{\leq p}(bw,a)$. Then

$$\begin{aligned} & f_{p,n+1,h,k}(wa) \\ = \; & \max\{\mathtt{r}_{\leq p}(vbwa) \mid |H(vbwa)| = h, |vbwa| = n+1\} \\ = \; & \max\{\mathtt{r}_{\leq p}(vbw) + \mathtt{nr}_{\leq p}(vbw,a) \mid |H(vbwa)| = h, |vbwa| = n+1\} \\ = \; & \max\{\mathtt{r}_{\leq p}(vbw) + \mathtt{nr}_{\leq p}(bw,a) \mid |H(vbwa)| = h, |vbwa| = n+1\} \\ = \; & \max_{b \in A_\diamond}\{\max\{\mathtt{r}_{\leq p}(vbw) \mid |H(vbw)| = h, |vbw| = n\} + \mathtt{nr}_{\leq p}(bw,a)\} \\ = \; & \max_{b \in A_\diamond}\{f_{p,n,h,k}(bw) + \mathtt{nr}_{\leq p}(bw,a)\}. \end{aligned}$$

Let us now assume that $a = \diamond$. Then

$$\begin{aligned} & f_{p,n+1,h,k}(wa) \\ = \; & \max\{\mathtt{r}_{\leq p}(vbw) + \mathtt{nr}_{\leq p}(bw,a) \mid |H(vbwa)| = h, |vbwa| = n+1\} \\ = \; & \max_{b \in A_\diamond}\{\max\{\mathtt{r}_{\leq p}(vbw) \mid |H(vbw)| = h-1, |vbw| = n\} + \mathtt{nr}_{\leq p}(bw,a)\} \\ = \; & \max_{b \in A_\diamond}\{f_{p,n,h-1,k}(bw) + \mathtt{nr}_{\leq p}(bw,a)\}. \end{aligned}$$

□

6 Conclusion

We presented efficient algorithms for computing the occurrences of PR-square factors and subwords in any partial word, and in the process we described an algorithm for computing all PR-run occurrences. Given a partial word w of length n, the number of PR-square factor occurrences in w can be computed in $O(n^2 \log\log n)$ time and the number of PR-square subword occurrences in $O(n^2 \log n)$ time. Furthermore, all PR-run occurrences in w can be computed in $O(n^2 \log\log n)$ time. The running time analyses of our algorithms use analytic ideas from (algorithmic) number theory. Our algorithms report repetitions in blocks of squares and use the idea of large divisors of the word's length. They can be useful in the combinatorial analysis of repetitions in partial words where computer checks are often applied.

We also extended previous analyses on the number of occurrences of PR-runs in total words to partial words. We showed, in particular, a linear upper bound on the number of runs when the number of holes is constant. We also showed how to recursively count the exact number of microruns, those of small period. In [7], tight upper and lower bounds were established on the number of occurrences of PR-square factors and subwords in a given partial word of length n in some special cases, i.e., when the number of holes is constant, or $n/2$, or n minus a constant, or n; note that for a constant number of holes, at most a constant number of PR-square subwords can be introduced into a partial word, and hence the bounds are identical to the total word case. We need to further our understanding of the counting of repetitions in partial words, several questions being linked to the algorithms' analyses.

We suggest the following problems for future work. Given a partial word of a fixed length over an alphabet of a fixed size: (1) Count the number of occurrences of squares and runs (not necessarily primitively-rooted); (2) Analyze the time complexity of computing all the squares and runs (not necessarily primitively-rooted); (3) Develop algorithms for computing all *distinct* runs; (4) Find upper and lower bounds on the number of distinct runs; (5) Extend the recent work of Kolpakov [26] on repetitions, where he classified all runs in a total word as *primary* or *secondary*, where the primary repetitions determine all other repetitions.

Acknowledgements

Project sponsored by the National Security Agency under Grant Number H98230-15-1-0232. The United States Government is authorized to reproduce and distribute reprints notwithstanding any copyright notation herein. This manuscript is submitted for publication with the understanding that the United States Government is authorized to reproduce and distribute reprints. This material is based upon work supported by the National Science Foundation under Grant No. DMS–1060775.

Thanks to Maxime Crochemore for suggesting that we look into runs in partial words and Sinziana Munteanu from Carnegie Mellon University for her help in refining the ideas in this paper. Thanks also to the referees of preliminary versions of this paper for their very valuable comments and suggestions.

A research assignment from the University of North Carolina at Greensboro for the first author is also gratefully acknowledged. Some of this assignment was spent at the IRIF: Institut de Recherche en Informatique Fondamentale of Université Paris-Diderot–Paris 7, France.

References

[1] K. Abrahamson. Generalized string matching. *SIAM Journal on Computing*, 16:1039–1051, 1987.

[2] A. Apostolico and F. P. Preparata. Optimal off-line detection of repetitions in a string. *Theoretical Computer Science*, 22:297–315, 1983.

[3] E. Bach and J. Shallit. *Algorithmic Number Theory, Vol. 1: Efficient Algorithms.* MIT Press, Cambridge, MA, 1996.

[4] H. Bannai, Tomohiro I, S. Inenaga, Y. Nakashima, M. Takeda, and K. Tsuruta. The "runs" theorem. arXiv:1406.0263v6 [cs.DM] 12 Jan 2015.

[5] H. Bannai, Tomohiro I, S. Inenaga, Y. Nakashima, M. Takeda, and K. Tsuruta. A new characterization of maximal repetitions by Lyndon trees. In *SODA 2015*, pages 562–571, 2015.

[6] F. Blanchet-Sadri. *Algorithmic Combinatorics on Partial Words.* Chapman & Hall/CRC Press, Boca Raton, FL, 2008.

[7] F. Blanchet-Sadri, M. Bodnar, J. Nikkel, J. D. Quigley, and X. Zhang. Squares and primitivity in partial words. *Discrete Applied Mathematics*, 185:26–37, 2015.

[8] F. Blanchet-Sadri, Y. Jiao, J. Machacek, J. D. Quigley, and X. Zhang. Squares in partial words. *Theoretical Computer Science*, 530:42–57, 2014.

[9] F. Blanchet-Sadri, R. Mercaş, A. Rashin, and E. Willett. Periodicity algorithms and a conjecture on overlaps in partial words. *Theoretical Computer Science*, 443:35–45, 2012.

[10] F. Blanchet-Sadri, J. Nikkel, J. D. Quigley, and X. Zhang. Computing primitively-rooted squares and runs in partial words. In J. Kratochvíl et al., editors, *IWOCA 2014, 25th International Workshop on Combinatorial Algorithms*, volume 8986 of *Lecture Notes in Computer Science*, pages 86–97, 2015. Springer International Publishing Switzerland.

[11] M. Crochemore. An optimal algorithm for computing the repetitions in a string. *Information Processing Letters*, 12:244–250, 1981.

[12] M. Crochemore, C. Hancart, and T. Lecroq. *Algorithms on Strings*. Cambridge University Press, 2007.

[13] M. Crochemore and L. Ilie. Maximal repetitions in strings. *Journal of Computer and System Sciences*, 74:796–807, 2008.

[14] M. Crochemore, L. Ilie, and W. Rytter. Repetitions in strings: Algorithms and combinatorics. *Theoretical Computer Science*, 410:5227–5235, 2009.

[15] M. Crochemore, L. Ilie, and L. Tinta. The "runs" conjecture. *Theoretical Computer Science*, 412:2931–2941, 2011.

[16] M. Crochemore and R. Mercaş. Fewer runs than word length. arXiv:1412.4646v1 [cs.DM] 15 Dec 2014.

[17] A. Diaconu, F. Manea, and C. Tiseanu. Combinatorial queries and updates on partial words. In M. Kutylowski, M. Gebala, and W. Charatonik, editors, *FCT 2009, 17th International Symposium on Fundamentals of Computation Theory*, volume 5699 of *Lecture Notes in Computer Science*, pages 96–108, Berlin, Heidelberg, 2009. Springer-Verlag.

[18] M. Fischer and M. Paterson. String matching and other products. In R. Karp, editor, *7th SIAM-AMS Complexity of Computation*, pages 113–125, 1974.

[19] F. Franek, R. Simpson, and W. F. Smyth. The maximum number of runs in a string. AWOCA 2003, 14th Australasian Workshop on Combinatorial Algorithms, pages 26–35, 2003.

[20] F. Franek and Q. Yang. An asymptotic lower bound for the maximal-number-of-runs functions. PSC 2006, Prague Stringology Conference, pages 3–8, 2006.

[21] F. Franek and Q. Yang. An asymptotic lower bound for the maximal number of runs in a string. *International Journal of Foundations of Computer Science*, 19:195–203, 2008.

[22] M. Giraud. Not so many runs in strings. In *LATA 2008, 2nd International Conference on Language and Automata Theory and Applications*, Lecture Notes in Computer Science, pages 245–252, Berlin, Heidelberg, 2008. Springer-Verlag.

[23] M. Giraud. Asymptotic behavior of the numbers of runs and microruns. *Information and Computation*, 207:1221–1228, 2009.

[24] V. Halava, T. Harju, and T. Kärki. On the number of squares in partial words. *RAIRO-Theoretical Informatics and Applications*, 44:125–138, 2010.

[25] G. H. Hardy and E. M. Wright. *An Introduction to the Theory of Numbers.* Oxford University Press, 2008.

[26] R. Kolpakov. On primary and secondary repetitions in words. *Theoretical Computer Science*, 418:71–81, 2012.

[27] R. Kolpakov and G. Kucherov. Finding maximal repetitions in a string in linear time. In *FOCS 1999, 40th Annual Symposium on Foundations of Computer Science*, pages 596–604, Los Alamitos, 1999. IEEE Computer Society Press.

[28] R. Kolpakov and G. Kucherov. On maximal repetitions in words. *Journal on Discrete Algorithms*, 1:159–186, 2000.

[29] M. Lothaire. *Combinatorics on Words.* Cambridge University Press, Cambridge, 1997.

[30] M. Lothaire. *Applied Combinatorics on Words.* Cambridge University Press, Cambridge, 2005.

[31] M. G. Main and R. J. Lorentz. An O(nlog n) algorithm for finding all repetitions in a string. *Journal of Algorithms*, 5:422–432, 1984.

[32] F. Manea, R. Mercaş, and C. Tiseanu. Periodicity algorithms for partial words. In F. Murlak and P. Sankowski, editors, *MFCS 2011, 36th International Symposium on Mathematical Foundations of Computer Science*, volume 6907 of *Lecture Notes in Computer Science*, pages 472–484, Berlin, Heidelberg, 2011. Springer-Verlag.

[33] W. Matsubara, K. Kusano, A. Ishino, H. Bannai, and A. Shinohara. New lower bounds for the maximum number of runs in a string. In *PSC 2008, Prague Stringology Conference*, pages 140–145, 2008.

[34] S. J. Puglisi, J. Simpson, and W. F. Smyth. How many runs can a string contain? *Theoretical Computer Science*, 401:165–171, 2008.

[35] W. Rytter. The number of runs in a string. *Information and Computation*, 205:459–1469, 2007.

[36] J. Simpson. Modified padovan words and the maximum number of runs in a word. *Australasian Journal of Combinatorics*, 46:129–145, 2010.