


The Number of Repetitions in 2D-Strings

Panagiotis Charalampopoulos 

Department of Informatics, King's College London, UK
Institute of Informatics, University of Warsaw, Poland
panagiotis.charalampopoulos@kcl.ac.uk

Jakub Radoszewski 


Institute of Informatics, University of Warsaw, Poland
Samsung R&D Poland, Warsaw, Poland
jrad@mimuw.edu.pl

Wojciech Rytter 

Institute of Informatics, University of Warsaw, Poland
rytter@mimuw.edu.pl

Tomasz Waleń 

Institute of Informatics, University of Warsaw, Poland
walen@mimuw.edu.pl

Wiktor Zuba 

Institute of Informatics, University of Warsaw, Poland
w.zuba@mimuw.edu.pl

Abstract

The notions of periodicity and repetitions in strings, and hence these of runs and squares, naturally extend to two-dimensional strings. We consider two types of repetitions in 2D-strings: *2D-runs* and *quartics* (quartics are a 2D-version of squares in standard strings). Amir et al. introduced 2D-runs, showed that there are $\mathcal{O}(n^3)$ of them in an $n \times n$ 2D-string and presented a simple construction giving a lower bound of $\Omega(n^2)$ for their number (*Theoretical Computer Science*, 2020). We make a significant step towards closing the gap between these bounds by showing that the number of 2D-runs in an $n \times n$ 2D-string is $\mathcal{O}(n^2 \log^2 n)$. In particular, our bound implies that the $\mathcal{O}(n^2 \log n + \text{output})$ run-time of the algorithm of Amir et al. for computing 2D-runs is also $\mathcal{O}(n^2 \log^2 n)$. We expect this result to allow for exploiting 2D-runs algorithmically in the area of 2D pattern matching.

A quartic is a 2D-string composed of 2×2 identical blocks (2D-strings) that was introduced by Apostolico and Brimkov (*Theoretical Computer Science*, 2000), where by quartics they meant only *primitively rooted* quartics, i.e. built of a primitive block. Here our notion of quartics is more general and analogous to that of squares in 1D-strings. Apostolico and Brimkov showed that there are $\mathcal{O}(n^2 \log^2 n)$ occurrences of primitively rooted quartics in an $n \times n$ 2D-string and that this bound is attainable. Consequently the number of distinct primitively rooted quartics is $\mathcal{O}(n^2 \log^2 n)$. The straightforward bound for the maximal number of distinct general quartics is $\mathcal{O}(n^4)$. Here, we prove that the number of distinct general quartics is also $\mathcal{O}(n^2 \log^2 n)$. This extends the rich combinatorial study of the number of distinct squares in a 1D-string, that was initiated by Fraenkel and Simpson (*Journal of Combinatorial Theory, Series A*, 1998), to two dimensions.

Finally, we show some algorithmic applications of 2D-runs. Specifically, we present algorithms for computing all occurrences of primitively rooted quartics and counting all general distinct quartics in $\mathcal{O}(n^2 \log^2 n)$ time, which is quasi-linear with respect to the size of the input. The former algorithm is optimal due to the lower bound of Apostolico and Brimkov. The latter can be seen as a continuation of works on enumeration of distinct squares in 1D-strings using runs (Crochemore et al., *Theoretical Computer Science*, 2014). However, the methods used in 2D are different because of different properties of 2D-runs and quartics.

2012 ACM Subject Classification Theory of computation → Pattern matching

Keywords and phrases 2D-run, quartic, run, square

Digital Object Identifier 10.4230/LIPIcs.ESA.2020.32



© Panagiotis Charalampopoulos, Jakub Radoszewski, Wojciech Rytter, Tomasz Waleń, and Wiktor Zuba;

licensed under Creative Commons License CC-BY

28th Annual European Symposium on Algorithms (ESA 2020).

Editors: Fabrizio Grandoni, Grzegorz Herman, and Peter Sanders; Article No. 32; pp. 32:1–32:18



Leibniz International Proceedings in Informatics

LIPICs Schloss Dagstuhl – Leibniz-Zentrum für Informatik, Dagstuhl Publishing, Germany

Funding *Panagiotis Charalampopoulos*: Partially supported by ERC grant TOTAL under the EU’s Horizon 2020 Research and Innovation Programme (agreement no. 677651).

Jakub Radoszewski: Supported by the Polish National Science Center, grant no. 2018/31/D/ST6/03991.

Tomasz Waleń: Supported by the Polish National Science Center, grant no. 2018/31/D/ST6/03991.

Wiktor Zuba: Supported by the Polish National Science Center, grant no. 2018/31/D/ST6/03991.

1 Introduction

Periodicity is one of the main and most elegant notions in stringology. It has been studied extensively both from the combinatorial and the algorithmic perspective; see e.g. the books [18, 25, 39]. A classic combinatorial result is the periodicity lemma due to Fine and Wilf [27]. From the algorithmic side, periodicity often poses challenges in pattern matching, due to the following fact: a pattern P can have many occurrences in a text T that are “close” to each other if and only if P has a “small” period. On the other hand, the periodic structure indeed allows us to overcome such challenges; see [18, 25].

Runs, also known as maximal repetitions, are a fundamental notion in stringology. A run is a periodic fragment of the text that cannot be extended without changing the period. Runs were introduced in [35]. Kolpakov and Kucherov presented an algorithm to compute all runs in a string in time linear with respect to the length of the string over a linearly-sortable alphabet [38]. Runs fully capture the periodicity of the underlying string and, since the publication of the algorithm for their linear-time computation, they have assumed a central role in algorithm design for strings. They have been exploited for text indexing [36], answering internal pattern matching queries in texts [16, 37], or reporting repetitions in a string [2, 15, 22], to name a few applications.

Kolpakov and Kucherov also posed the so-called runs conjecture which states that there are at most n runs in a string of length n . A long line of work on the upper [19, 20, 21, 31, 42, 43, 44] and lower bounds [30, 41, 45] was concluded by Bannai et al. who positively resolved the runs conjecture in [10] (see also an alternative proof in [23] and a tighter upper bound for binary strings from [28]).

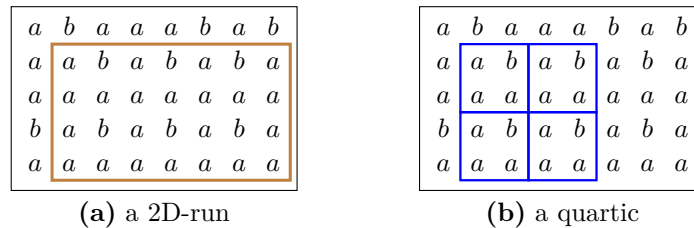
A square is a concatenation of two copies of the same string. Fraenkel and Simpson [29] showed that a string of length n contains at most $2n$ distinct square factors. This bound was improved in [26, 34]. All distinct squares in a string of length n can be computed in $\mathcal{O}(n)$ time assuming an integer alphabet [11, 22, 33] (see [46] for an earlier $\mathcal{O}(n \log n)$ algorithm).

Pattern matching and combinatorics on 2D strings have been studied for more than 40 years, see e.g. [1, 4, 9, 14, 18, 25]. In this paper we consider 2-dimensional versions of runs, introduced by Amir et al. [5, 6], and of repetitions in 2D-strings, introduced by Apostolico and Brimkov [7]. As discussed in [6, 8], one could potentially exploit such repetitions in a 2D-string, which could for instance be an image, in order to compress it.

A *2D-run* in a 2D-string A is a subarray of A that is both horizontally periodic and vertically periodic and that cannot be extended by a row or column without changing the horizontal or vertical periodicity (a formal definition follows in Section 2); see Figure 1(a). Amir et al. [5, 6] have shown that the maximum number of 2D-runs in an $n \times n$ array is $\mathcal{O}(n^3)$ and presented an example with $\Theta(n^2)$ 2D-runs. In [6] they presented an $\mathcal{O}(n^2 \log n + \text{output})$ -time algorithm for computing 2D-runs.

A *quartic* is a configuration that is composed of 2×2 occurrences of an array W (see Figure 1(b)) and a *tandem* is a configuration consisting of two occurrences of an array W that share one side (Apostolico and Brimkov [7] also considered another type of tandems, which

share one corner; see also [3]). An array W is called *primitive* if it cannot be partitioned into non-overlapping replicas of some array W' . Apostolico and Brimkov [7] considered only quartics and tandems with primitive W (we call them *primitively rooted*) and showed tight asymptotic bounds $\Theta(n^2 \log^2 n)$ and $\Theta(n^3 \log n)$ for the maximum number of occurrences of such quartics and tandems in an $n \times n$ array, respectively. In [8] they presented an optimal $\mathcal{O}(n^3 \log n)$ -time algorithm for computing all *occurrences* of tandems with primitive W . This extends a result that a 1D-string of length n contains $\mathcal{O}(n \log n)$ occurrences of primitively rooted squares and they can all be computed in $\mathcal{O}(n \log n)$ time; see [17, 46]. In this paper we consider the numbers of *all distinct* quartics, which is a more complicated problem.



■ **Figure 1** Examples of a 2D-run and a quartic.

When computing 2D-runs we consider positioned runs: two 2D-runs with same content but starting in different points are considered distinct. However in case of quartics, similarly as in case of 1D-squares, we consider unpositioned quartics; if two quartics have the same content but start in different positions, we consider them equal.

Our Results.

- We show that the number of 2D-runs in an $n \times n$ array is $\mathcal{O}(n^2 \log^2 n)$. This improves upon the $\mathcal{O}(n^3)$ upper bound of Amir et al. [5, 6] and proves that their algorithm computes all 2D-runs in an $n \times n$ 2D-string in $\mathcal{O}(n^2 \log^2 n)$ time (**Section 3**).
- We show that the number of distinct quartics in an $n \times n$ array is $\mathcal{O}(n^2 \log^2 n)$. This can be viewed as an extension of the bounds on the maximum number of distinct square factors in a 1D-string [26, 29] (**Section 4**).
- We present algorithmic implications of the new upper bound for 2D-runs. We show that all occurrences of primitively rooted quartics can be computed in quasi-linear, $\mathcal{O}(n^2 \log^2 n)$ time, which is optimal by the bound of Apostolico and Brimkov [7]. Thus our algorithm complements the result of Apostolico and Brimkov [8] who gave an optimal algorithm for computing all occurrences of primitively rooted tandems. We also show that all distinct quartics can be computed in quasi-linear, $\mathcal{O}(n^2 \log^2 n)$ time, which extends efficient computation of distinct squares in 1D-strings [11, 22, 33] to 2D (**Section 5**).
- As an easy side result, we show tight $\Theta(n^3)$ bounds for the maximum number of distinct tandems in an $n \times n$ array and how to report them in $\mathcal{O}(n^3)$ time (**Section 2**).

2 Preliminaries

1D-Strings. We denote by $[a, b]$ the set $\{i \in \mathbb{Z} : a \leq i \leq b\}$. Let $S = S[1]S[2] \cdots S[|S|]$ be a *string* of length $|S|$ over an alphabet Σ . The elements of Σ are called *letters*. For two positions i and j on S , we denote by $S[i..j] = S[i] \cdots S[j]$ the *fragment* of S that starts at position i and ends at position j (it equals ε if $j < i$). A positive integer p is called a *period* of S if $S[i] = S[i + p]$ for all $i = 1, \dots, |S| - p$. We refer to the smallest period as *the period* of the string, and denote it by $\text{per}(S)$.

► **Lemma 1** (Periodicity Lemma (weak version), Fine and Wilf [27]). *If p and q are periods of a string S and satisfy $p + q \leq |S|$, then $\gcd(p, q)$ is also a period of S .*

A string S is called *periodic* if $\text{per}(S) \leq |S|/2$. By ST and S^k we denote the concatenation of strings S and T and k copies of the string S , respectively. A string S is called *primitive* if it cannot be expressed as U^k for a string U and an integer $k > 1$.

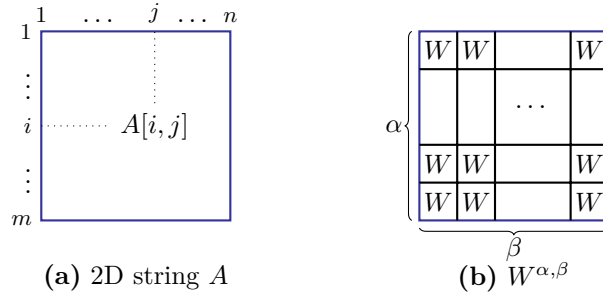
A string of the form U^2 for string U is called a *square*. A square U^2 is called *primitively rooted* if U is primitive. We will make use of the following important property of squares.

► **Lemma 2** (Three Squares Lemma, [24]). *Let U, V and W be three strings such that U^2 is a proper prefix of V^2 , V^2 is a proper prefix of W^2 and U is primitive. Then $|U| + |V| \leq |W|$.*

A *run* (also known as *maximal repetition*) in S is a periodic fragment $R = S[i..j]$ which cannot be extended either to the left or to the right without increasing the period $p = \text{per}(R)$, i.e. if $i > 1$ then $S[i - 1] \neq S[i + p - 1]$ and if $j < |S|$ then $S[j + 1] \neq S[j - p + 1]$. Let $\mathcal{R}(S)$ denote the set of all runs of string S . For periodic fragment $U = S[a..b]$, the run that extends U is the unique run $R = S[i..j]$ such that $i \leq a \leq b \leq j$ and $\text{per}(R) = \text{per}(U)$. An occurrence of a square U^2 is said to be *induced* by a run R if R extends U^2 . Every square is induced by exactly one run [22].

2D-Strings. Let A be an $m \times n$ array (2D-string). We denote the height and width of A by $\text{height}(A) = m$ and $\text{width}(A) = n$, respectively. By $A[i, j]$ we denote the cell in the i th row and j th column of A ; see Figure 2(a). By $A[i_1..i_2, j_1..j_2]$ we denote the subarray formed of rows i_1, \dots, i_2 and columns j_1, \dots, j_2 .

A positive integer p is a *horizontal period* of A if the i -th column of A equals the $(i + p)$ -th column of A for all $i = 1, \dots, n - p$. We denote the smallest horizontal period of A by $\text{hper}(A)$. Similarly, a positive integer q is a *vertical period* of A if the i -th row of A equals the $(i + q)$ -th row of A for all $i = 1, \dots, m - q$; the smallest vertical period of A is denoted by $\text{vper}(A)$.



■ **Figure 2** A 2D-string and the structure of $W^{\alpha, \beta}$.

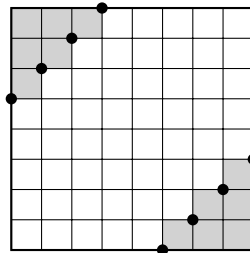
An $r \times c$ subarray $B = A[i_1..i_2, j_1..j_2]$ of A is a *2D-run* if $\text{hper}(B) \leq c/2$, $\text{vper}(B) \leq r/2$ and extending B by a row or column, i.e. either of $A[i_1 - 1, j_1..j_2]$, $A[i_2 + 1, j_1..j_2]$, $A[i_1..i_2, j_1 - 1]$, or $A[i_1..i_2, j_2 + 1]$, would result in a change of the smallest vertical or the horizontal period.

If W is a 2D array, then by $W^{\alpha, \beta}$ we denote an array that is composed of $\alpha \times \beta$ copies of W ; see Figure 2(b). A *tandem* of W is an array of the form $W^{1,2}$ and a *quartic* of W is the array $W^{2,2}$. A 2D array A is called *primitive* if $A = B^{\alpha, \beta}$ for positive integers α, β implies that $\alpha = \beta = 1$. The *primitive root* of an array A is the unique primitive array B for which $A = B^{\alpha, \beta}$ for $\alpha, \beta \geq 1$.

Apostolico and Brimkov [7] proved the following upper bound, and showed that it is tight by giving a corresponding lower bound.

► **Fact 3** (Lemma 5 in [7]). *A 2D array of size $n \times n$ has $\mathcal{O}(n^2 \log^2 n)$ occurrences of primitively rooted quartics.*

We say that a quartic $Q = W^{2,2}$ is *induced* by a 2D-run R if Q is a subarray of R and $\text{hper}(R)$ and $\text{vper}(R)$ divide the width and height of W , respectively.



■ **Figure 3** Shaded positions contain letters b , all the other the letters a . Each rectangle with top-left and bottom-right corners marked is a 2D-run; altogether there are 18 distinct 2D-runs, including two of the form $b^{2,2}$. There are also 10 distinct quartics $a^{\alpha,\beta}$, where $0 < \alpha, \beta \leq 8$ are even and $\alpha + \beta \leq 10$. There is also the quartic $b^{2,2}$ (altogether 11 distinct quartics). The centrally placed quartic $a^{2,2}$ is contained in 16 2D-runs. There are only two distinct primitively rooted quartics.

► **Observation 4.** *Every quartic is induced by a 2D-run. However; the same quartic can be induced even by $\Theta(n^2)$ 2D-runs; say the middle quartic $a^{2,2}$ in Figure 3.*

► **Remark 5.** The fact that a string of length n has $\mathcal{O}(n \log n)$ occurrences of primitively rooted squares immediately shows (by the fact that a square is induced by exactly one run) that it has $\mathcal{O}(n \log n)$ runs. However, an analogous argument applied for quartics and 2D-runs does not give a non-trivial upper bound for the number of the latter because of Observation 4.

In our algorithms, we use a variant of the Dictionary of Basic Factors in 2D (2D-DBF in short) that is similar to the one presented in [25]. Namely, to each subarray of A whose width and height is an integer power of 2 we assign an integer identifier from $[0, n^2]$ so that two arrays with the same dimensions are equal if and only if their identifiers are equal. The total number of such subarrays is $\mathcal{O}(n^2 \log^2 n)$ and the identifiers can be assigned in $\mathcal{O}(n^2 \log^2 n)$ time; see [25]. Using 2D-DBF, we can assign an identifier to a subarray of A of arbitrary dimensions $r \times c$ being a quadruple of 2D-DBF identifiers of its four $2^i \times 2^j$ subarrays that share one of its corners, where $2^i \leq r < 2^{i+1}$ and $2^j \leq c < 2^{j+1}$. Such quadruples preserve the property that two subarrays of the same dimensions are equal if and only if the 2D-DBF quadruples are the same.

As an illustration, we show a tight bound for the number of distinct tandems and an optimal algorithm for computing them.

► **Theorem 6.** *The maximum number of distinct tandems in an $n \times n$ array A is $\Theta(n^3)$. All distinct tandems in an $n \times n$ array can be reported in the optimal $\Theta(n^3)$ time.*

Proof. Let us fix two row numbers $i < i'$ in A . Then, the number of distinct tandems with top row i and bottom row i' is $\mathcal{O}(n)$ by the fact that a string of length n contains $\mathcal{O}(n)$ squares [26, 29]. Thus, in total there are $\mathcal{O}(n^3)$ distinct tandems. For the lower bound, let

the i th row of A be filled with occurrences of the letter i . Every subarray of A of even width is a tandem. For each distinct triplet of top and bottom rows and even width, we obtain a distinct tandem.

Let us proceed to the algorithm. For a height $h \in [1, n]$, we assign integer identifiers from $[1, n^2]$ that preserve lexicographical comparison to all height- h substrings of columns of A . They can be assigned using the generalized suffix tree [18, 47] of the columns of A in $\mathcal{O}(n^2 \log n)$ time. Let B_h be an array such that $B_h[i, j]$ stores the identifier of $A[i..i+h-1, j]$. To a subarray $W = A[i..i+h-1, j..j+w-1]$ we assign an *identifier* $\text{id}(W) = B_h[i, j..j+w-1]$. Then for any two subarrays W and W' of height h , $W = W'$ if and only if $\text{id}(W) = \text{id}(W')$. For every height $h = 1, \dots, n$ and row i , we find all distinct squares in $B_h[i, 1], \dots, B_h[i, n]$ in $\mathcal{O}(n)$ time [11, 22, 33]. This corresponds to the set of distinct tandems with top row i and bottom row $i+h-1$. Finally, we assign identifiers from 2D-DBF of A to each of the tandems and use radix sort to sort them and enumerate distinct tandems. ◀

3 Improved Upper Bound for 2D-Runs

We introduce the framework that Amir et al. used for efficiently computing 2D-runs [5, 6].

We say that a subarray $B = A[i_1..i_2, j_1..j_2]$ of A is a *horizontal run* if it is horizontally periodic (that is, $\text{hper}(B) \leq \text{width}(B)/2$) and extending B by either of the columns $A[i_1..i_2, j_1-1]$ or $A[i_1..i_2, j_2+1]$ would result in a change of the smallest horizontal period. (Note that B does not have to be vertically periodic.)

For $k \in [1, \lfloor \log n \rfloor]$ and $i \in [1, n-2^k+1]$, let H_i^k be the string obtained by replacing the columns of array $A[i..i+2^k-1, 1..n]$ with metasympols such that $H_i^k[j] = H_i^k[j']$ if and only if $A[i..i+2^k-1, j] = A[i..i+2^k-1, j']$. Notice that each such horizontal run of height 2^k corresponds to a run in some H_i^k .

The following lemma will enable us to “anchor” each 2D-run R in the top-left or bottom-left corner of a horizontal run of “similar” height as R . It was proved in [6], but we provide a proof for completeness.

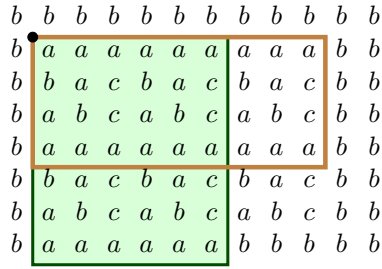
► **Lemma 7** (Lemma 7 in [6]). *Let R be a 2D-run whose height is in the range $[2^k, 2^{k+1})$. Then there is a horizontal run R' of height 2^k with $\text{hper}(R') = \text{hper}(R)$ and $\text{width}(R') \geq \text{width}(R)$ such that top-left or bottom-left corners of R and R' coincide (see Figure 4).*

Proof. Let $R = A[i_1..i_2, j_1..j_2]$ be the 2D-run in scope and let $k = \lfloor \log(i_2 - i_1 + 1) \rfloor$. We have to show that at least one of the two following statements holds.

- There is a run $R_1 = S[j_1..b]$ in $S = H_{i_1}^k$ with smallest period p and $b \geq j_2$.
- There is a run $R_2 = T[j_1..d]$ in $T = H_{i_2-2^k+1}^k$ with smallest period p and $d \geq j_2$.

Since $\text{vper}(R) \leq \text{height}(R)/2$, all distinct rows of R are represented in each of $U = S[j_1..j_2]$ and $V = T[j_1..j_2]$ and hence $p = \text{per}(U) = \text{per}(V)$. Let $R_1 = S[a..b]$ be the run that extends U and $R_2 = T[c..d]$ be the run that extends V . Let us suppose towards a contradiction that $\max(a, c) < j_1$. Then, $A[i_1..i_2, j_1-1] = A[i_1..i_2, j_1-1+p]$, which contradicts R being a run, since R and $B = A[i_1..i_2, j_1-1..j_2]$ have the same horizontal and vertical periods. ◀

The sum of the lengths of the runs in a string of length n can be $\Omega(n^2)$ as shown in [32]. However, we prove the following lemma, which is crucial for our approach. We will use it to obtain an overall bound on the possible widths of 2D-runs for our anchors.



■ **Figure 4** The shaded 7×6 subarray is a 2D-run R , with vertical period 3 and horizontal period $p = 3$. The other marked 4×9 rectangle encloses a horizontal run R' with the same top-left corner and the same horizontal period as R . We have $2 \cdot p \leq \text{width}(R) \leq \text{width}(R')$.

► **Lemma 8.** *For any string S of length n we have that*

$$\rho(S) := \sum_{R \in \mathcal{R}(S)} (|R| - 2 \cdot \text{per}(R) + 1) = \mathcal{O}(n \log n).$$

Proof. We consider for each run $R = S[i..j]$ of S the interval $I_R = [i, j - 2 \cdot \text{per}(R) + 1]$. Note that $\rho(S) = \sum_{R \in \mathcal{R}(S)} |I_R|$.

Observe that for every $a \in I_R$ the string $S[a..a + \text{per}(R) - 1]$ is primitive, since if it was of the form U^k for a string U and an integer $k > 1$, then $|U| < \text{per}(R)$ would be a period of R , a contradiction. Hence, at each position $a \in I_R$ there is an occurrence of a primitively rooted square of length $2 \cdot \text{per}(R)$.

A direct application of the Three Squares Lemma (Lemma 2) implies that at most $\mathcal{O}(\log n)$ primitively rooted squares can start at each position a . Each such square extends to a unique run. Thus, each position i belongs to $\mathcal{O}(\log n)$ intervals I_R for $R \in \mathcal{R}(S)$. This completes the proof. ◀

We are now ready to prove the main result of this section.

► **Theorem 9.** *There are $\mathcal{O}(n^2 \log^2 n)$ 2D-runs in an $n \times n$ array A .*

Proof. We will iterate over all horizontal runs $R' = A[i..i', j..j']$ whose height is a power of 2, i.e. $i' = i + 2^k - 1$ for some k . For each such horizontal run R' , we consider the 2D-runs R with:

- (a) top-left corner $A[i, j]$ or bottom-left corner $A[i', j]$,
- (b) $\text{hper}(R) = \text{hper}(R')$, and
- (c) $\text{height}(R) \in [2^k, 2^{k+1})$.

For each such 2D-run R , we have $\text{width}(R) \in [2 \cdot \text{hper}(R'), \text{width}(R')]$, else the horizontal period would break, i.e. property (b) would be violated. Let us notice that R' corresponds to a run $U = H_i^k[j..j'] \in \mathcal{R}(H_i^k)$. In particular, $\text{width}(R) \in [2 \cdot \text{per}(U), |U|]$.

Lemma 7 implies that each 2D-run is accounted for at least once in this manner. It is thus enough to bound the number of considered runs. We have n choices for i and $\log n$ choices for k . Further, due to Lemma 8, for each corresponding meta-string H_i^k we have $\mathcal{O}(n \log n)$ choices for a pair (j, c) such that $U = H_i^k[j..j'] \in \mathcal{R}(H_i^k)$ and $c \in [2 \cdot \text{per}(U), |U|]$. In total, we thus have $\mathcal{O}(n^2 \log^2 n)$ choices for (i, k, j, c) . We will complete the proof by showing that there is only a constant number of 2D-runs with top-left corner $A[i, j]$, width w and whose height is in the range $[2^k, 2^{k+1})$. (2D-runs with bottom-left corner $A[i', j]$ can be bounded symmetrically.)

▷ **Claim 10** (cf. Lemma 10 in [6]). Let B be an $r \times c$ array with $r \in [2^k, 2^{k+1})$. Then, there are at most two integers $p > 2^{k-1}$ such that $p = \text{vper}(B') \leq \text{height}(B')/2$ for B' consisting of the top $\text{height}(B') \geq 2^k$ rows of B .

Proof. Consider S to be the meta-string obtained by replacing the rows of B by single letters. Then, a direct application of the Three Squares Lemma (Lemma 2) to S yields the claimed bound. ◁

We apply Claim 10 to $B = A[i.. \min(i + 2^{k+1} - 2, n), j.. j + c - 1]$. If $\text{vper}(R) \leq 2^{k-1}$, then $\text{vper}(R) = \text{vper}(R')$ by the Periodicity Lemma (Lemma 1) applied to the meta-string obtained by replacing the rows of the intersection of R' and B by single letters. Now Claim 10 implies that there are at most three choices to make for the vertical period: $\text{vper}(R')$ and the two integers from the claim. Finally, for fixed top-left corner, width and vertical period we can have a single 2D-run. This concludes the proof. ◀

Amir et al. [6] presented the following algorithmic result.

► **Theorem 11** ([6]). *All 2D-runs in an $n \times n$ array can be computed in $\mathcal{O}(n^2 \log n + \text{output})$ time, where *output* is the number of 2D-runs reported.*

By combining Theorems 9 and 11 we get the following corollary.

► **Corollary 12.** *All 2D-runs in an $n \times n$ array can be computed in $\mathcal{O}(n^2 \log^2 n)$ time.*

4 Upper Bound on the Number of Distinct Quartics

Fact 3 that originates from [7] shows that an $n \times n$ array A has $\mathcal{O}(n^2 \log^2 n)$ occurrences of primitively rooted quartics. This obviously implies that the number of distinct primitively rooted quartics is upper bounded by $\mathcal{O}(n^2 \log^2 n)$. Unfortunately, an array can contain $\Theta(n^4)$ occurrences of general quartics; this takes place e.g. for a unary array. In this section we show that $\mathcal{O}(n^2 \log^2 n)$ is also an upper bound for the number of *distinct* general quartics, i.e. subarrays of A of the form $W^{\alpha, \beta}$ for even $\alpha, \beta \geq 2$ and primitive W .

The following lemma and its corollary are the combinatorial foundation of our proofs. An array W with $\text{height}(W) \in [2^a, 2^{a+1})$ and $\text{width}(W) \in [2^b, 2^{b+1})$ will be called an (a, b) -array.

► **Lemma 13.** *Let a, b be non-negative integers and W, W' be different primitive (a, b) -arrays. If occurrences of $W^{2,3}$ and $(W')^{2,3}$ (of $W^{3,2}$ and $(W')^{3,2}$, respectively) in A share the same corner (i.e., top-left, top-right, bottom-left or bottom-right), then $\text{width}(W) = \text{width}(W')$ ($\text{height}(W) = \text{height}(W')$, respectively).*

Proof. Clearly it is sufficient to prove the lemma for $W^{2,3}$ and $(W')^{2,3}$. Assume w.l.o.g. that occurrences of $W^{2,3}$ and $(W')^{2,3}$ in A share the top-left corner and consider their overlap X .

Each of the rows of X has periods $\text{width}(W)$ and $\text{width}(W')$. Assume w.l.o.g. that $\text{width}(W) \leq \text{width}(W')$. Then

$$\text{width}(X) = 3 \cdot \text{width}(W) \geq \text{width}(W) + 2^{a+1} \geq \text{width}(W) + \text{width}(W').$$

By the Periodicity Lemma (Lemma 1), $p = \text{gcd}(\text{width}(W), \text{width}(W'))$ is a horizontal period of X .

The array X contains at least one occurrence of W and W' in its top-left corner. Hence, W and W' have a horizontal period p . If $\text{width}(W) < \text{width}(W')$, then $\text{width}(W')$ cannot be a multiple of $\text{width}(W)$, because then we would have $\text{width}(W') > 2^{a+1}$. Hence, if $\text{width}(W) < \text{width}(W')$, we would have $p < \text{width}(W)$ which by $p \mid \text{width}(W)$ would mean that W is not primitive. This indeed shows that $\text{width}(W) = \text{width}(W')$. ◀

► **Corollary 14.** *Let a, b be non-negative integers and W, W' be different (a, b) -arrays. If occurrences of $W^{3,3}$ and $(W')^{3,3}$ in A share the same corner (i.e., top-left, top-right, bottom-left or bottom-right), then at least one of W, W' is not primitive.*

If $V^{2,2}$ is a non-primitively rooted quartic, then there exists a primitive array W such that $V = W^{\alpha,\beta}$ and at least one of α, β is greater than one. We will call the quartic $W^{2\alpha,2\beta}$ *thin* if $\alpha = 1$ or $\beta = 1$ for this decomposition, and *thick* otherwise. We refer to *points* in A as the $(n+1)^2$ positions where row and column delimiters intersect. Let us first bound the number of distinct thin quartics. For $\beta > 1$, we consider any rightmost occurrence of every such quartic, that is, any occurrence $A[i_1 \dots i_2, j_1 \dots j_2]$ that maximizes j_1 .

► **Lemma 15.** *The total number of distinct thin quartics in A is $\mathcal{O}(n^2 \log^2 n)$.*

Proof. We give a proof for quartics of the form $W^{2,2\beta}$ for primitive W and $\beta > 1$; the proof for quartics of the form $W^{2\alpha,2}$ for $\alpha > 1$ is symmetric. We consider each pair of positive integers a, b and show that each point holds the top-left corner of at most two rightmost occurrences of $W^{2,2\beta}$ for primitive (a, b) -arrays W and $\beta > 1$.

Assume to the contrary that the rightmost occurrences of $W^{2,2\beta}$, $(W')^{2,2\beta'}$ and $(W'')^{2,2\beta''}$ share their top-left corner for primitive (a, b) -arrays W, W', W'' . The arrays W, W', W'' are pairwise different, since otherwise one of the occurrences would not be the rightmost. By Lemma 13, we have $\text{width}(W) = \text{width}(W') = \text{width}(W'')$. Assume w.l.o.g. that $\text{height}(W) < \text{height}(W') < \text{height}(W'')$.

Let (i, j) denote the top-left corner of the three quartics. Let us consider three length- 2ℓ strings formed of metacharacters that correspond to row fragments:

$$(A[i, j \dots j + w - 1]), \dots, (A[i + 2\ell - 1, j \dots j + w - 1])$$

for $w = \text{width}(W)$ and $\ell \in \{\text{height}(W), \text{height}(W'), \text{height}(W'')\}$. All the three strings need to be primitively rooted squares. We apply the Three Squares Lemma (Lemma 2) to conclude that $\text{height}(W'') > \text{height}(W) + \text{height}(W') > 2^{a+1}$, a contradiction. ◀

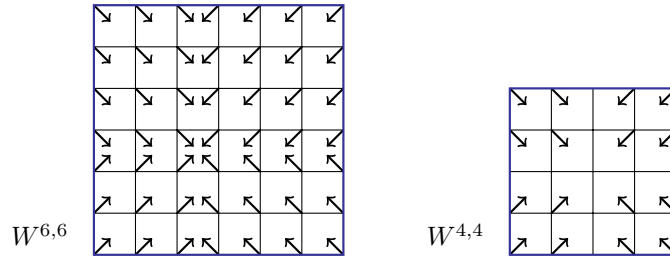
Now let us proceed to thick quartics. Unfortunately, in this case a single point can be the top-left corner of a linear number of rightmost occurrences of thick quartics; see the example in Figure 3. Let us consider an occurrence of $W^{\alpha,\beta}$ for even $\alpha, \beta > 2$ and primitive W , called a *positioned quartic*. It implies $\alpha \cdot \beta$ occurrences of W . Let us call all corners of all these occurrences of W *special points* of this positioned quartic. Each special point stores a direction in $\{\text{top-left}, \text{top-right}, \text{bottom-left}, \text{bottom-right}\}$. A special point has one of the directions if it is the respective corner of an occurrence of $W^{3,3}$ in this positioned quartic. Clearly, since $\alpha, \beta \geq 4$, for every special point in $W^{\alpha,\beta}$ except for the middle row if $\alpha = 4$ or middle column if $\beta = 4$, one can assign such a direction (if many directions are possible, we choose an arbitrary one); see Figure 5.

The quartics with primitive root W are called *W-quartics*. The set of all special points (with directions) of *all* positioned thick *W*-quartics for a given W is denoted by $\text{SpecialPoints}(W)$. Among *W*-quartics of the same height we distinguish the ones with maximal width, which we call *h-maximal* (horizontally maximal). Let us observe that each *W*-quartic is contained in an occurrence of some *h-maximal W*-quartic.

► **Theorem 16.** *The number of distinct quartics in an $n \times n$ array is $\mathcal{O}(n^2 \log^2 n)$.*

Proof. By Fact 3 and Lemma 15 it suffices to show that the total number of distinct thick quartics in A is $\mathcal{O}(n^2 \log^2 n)$. Let us fix non-negative integers a, b . It is enough to show that the number of distinct subarrays of A of the form $W^{\alpha,\beta}$ for even $\alpha, \beta > 2$ and any primitive (a, b) -array W is $\mathcal{O}(n^2)$.

The sets of special points have the following properties. Claim 17 follows from Corollary 14.



■ **Figure 5** Special points of a positioned quartic with primitive root W with associated directions of four types. The arrow indicates the corner (four possibilities) of $W^{3,3}$ which is contained in the quartic. If several assignments of directions are possible, only one of them is chosen (it does not matter which one). In case of $W^{4,4}$ the middle row and column are not special.

▷ **Claim 17.** For primitive (a, b) -arrays $W \neq W'$, $SpecialPoints(W) \cap SpecialPoints(W') = \emptyset$.

For an array W , let us denote by $ThickQuartics(W)$ the total number of thick quartics in A with primitive root W .

▷ **Claim 18.** For a primitive (a, b) -array W , $ThickQuartics(W) < |SpecialPoints(W)|$.

Proof. For each $\alpha = 4, 6, \dots$ in this order, we select one positioned h-maximal W -quartic U_α of height $\alpha \cdot \text{height}(W)$. The number of distinct W -quartics in A of height $\alpha \cdot \text{height}(W)$ is at most the number of special points in U_α in any of its rows. Note that this statement also holds if $U_\alpha = W^{\alpha,4}$; then there are still four special points in each (non-middle if $\alpha = 4$) row.

We describe a process of assigning distinct W -quartics to distinct special points in $SpecialPoints(W)$. Assume all points in this set are initially not marked. We choose any single row from U_α with all special points in this row still not marked. Then we mark all these special points. We can always choose a suitable row because the heights are increasing.

This way each W -quartic is assigned to only one special point from $SpecialPoints(W)$. ◁

By the claims, the total number of thick W -quartics for primitive (a, b) -arrays W is bounded by:

$$\sum_W ThickQuartics(W) < \sum_W |SpecialPoints(W)| \leq 4(n+1)^2,$$

where the sum is over all primitive (a, b) -arrays W . The conclusion follows. ◀

5 Algorithms for Computing Quartics

In this section we show algorithmic applications of 2D-runs related to quartics.

► **Theorem 19.** All occurrences of primitively rooted quartics in an $n \times n$ array A can be computed in the optimal $\mathcal{O}(n^2 \log^2 n)$ time.

Proof. Let us consider a 2D-run $R = A[i_1 \dots i_2, j_1 \dots j_2]$ with periods $\text{hper}(R) = p$ and $\text{vper}(R) = q$. It induces primitively rooted quartics of width $2p$ and height $2q$. The set of top-left corners of these quartics forms a rectangle $\hat{R} = [i_1, i_2 - 2p + 1] \times [j_1, j_2 - 2q + 1]$. We denote by $\mathcal{F}_{p,q}$ the family of such rectangles \hat{R} over 2D-runs R with the same periods p, q .

Such rectangles for different 2D-runs may overlap, even when the dimensions of the quartic are fixed (see Observation 4). In order not to report the same occurrence multiple times, we need to compute, for every dimensions of a quartic, all points in the union of

the corresponding rectangles. This could be done with an additional $\log n$ -factor in the complexity using a standard line sweep algorithm [12]. However, we can achieve $\mathcal{O}(n^2 \log^2 n)$ total time using the fact that the total number of occurrences reported is $\mathcal{O}(n^2 \log^2 n)$.

▷ **Claim 20.** Let $\mathcal{F}_1, \dots, \mathcal{F}_k$ be families of 2D rectangles in $[1, n]^2$ and let $r = \sum_{i=1}^k |\mathcal{F}_i|$. We can compute k (not necessarily disjoint) sets of grid points $Out_i = \bigcup \mathcal{F}_i$ in $\mathcal{O}(n + r + \text{output})$ total time, where $\text{output} = \sum_i |Out_i|$ is the total number of reported points.

Proof. We design an efficient line sweep algorithm. We will perform a separate line sweep, left to right, for each family \mathcal{F}_i .

The sweep goes over horizontal (x) coordinates in a left-to-right manner. The broom stores vertical (y) coordinates of horizontal sides of rectangles that it currently intersects. They are stored in a sorted list L of pairs (y, c) , where y is the coordinate, and c is the count of rectangles with bottom side at coordinate y minus the count of the rectangles with top side at coordinate y . Only pairs with non-zero second component are stored. Clearly, the second components of the list elements always sum up to 0.

A coordinate x is processed if L is non-empty before accessing it or there exist any vertical sides of rectangles at x . All vertical sides with the same y -coordinate are processed in a batch. For every such batch we want to guarantee that endpoints of all sides are stored in a list B in a top-down order.

A top (bottom) endpoint at vertical coordinate y is stored as $(y, +1)$ ($(y, -1)$, respectively).

Let us now describe how to process a horizontal coordinate x . Let us merge the list L that is currently in the broom with the list B of the batch by the first components. If there is more than one pair with the same first component, we merge all of them together, summing up the second components.

Let us denote by L' the resulting list. We iterate over all elements of L' , keeping track of the partial sum of second components, denoted as s . For every element (y, c) of L' , the point (x, y) is reported for $\bigcup \mathcal{F}_i$. Moreover, if the partial sum s before considering c was positive and the previous element of L' is (y', c') , all points $(x, y' + 1), \dots, (x, y - 1)$ are reported to Out_i .

Finally, all pairs with second component equal to zero are removed from L' which becomes the new list L .

Let us now analyze the complexity of the algorithm. The line sweep makes n steps. The total size of lists B across all families \mathcal{F}_i is $\mathcal{O}(r)$ and they can be constructed simultaneously in $\mathcal{O}(n + r)$ time via bucket sort.

Processing a batch with list B takes $\mathcal{O}(|L| + |B|)$ time plus the time to report points in Out_i . As we have already noticed, the sum of $\mathcal{O}(|B|)$ components is $\mathcal{O}(r)$. For every element (y, c) of the initial list L , a point with the vertical coordinate y is reported upon merging; hence, the sum of $\mathcal{O}(|L|)$ components is dominated by $\mathcal{O}(\text{output})$. Overall we achieve time complexity $\mathcal{O}(n + r + \text{output})$. ◀

We apply the claim to the families $\mathcal{F}_{p,q}$. Then r and output are upper bounded by $\mathcal{O}(n^2 \log^2 n)$ by Theorem 9 and Fact 3, respectively. The optimality of our algorithm's complexity is due to the $\Omega(n^2 \log^2 n)$ lower bound on the maximum number of occurrences of primitively rooted quartics from [7]. ◀

We proceed to an efficient algorithm for enumerating distinct, not necessarily primitively rooted, quartics using 2D-runs. The solution for an analogous problem for 1-dimensional strings (computing distinct squares from runs) uses Lyndon roots of runs [22]. However, in 2 dimensions it is not clear if a similar approach could be applied efficiently, say, with the aid

32:12 The Number of Repetitions in 2D-Strings

of 2D Lyndon words [40] as Lyndon roots of 2D-runs. We develop a different approach in which the workhorse is the following auxiliary problem related to the folklore nearest smaller value problem.

Let us consider a grid of height m in which every cell can be black or white. We say that the grid forms a *staircase* if the set of white cells in each row is nonempty and is a prefix of this row (see Figure 6). A staircase can be uniquely determined by an array $Whites[1..m]$ such that $Whites[i]$ is the number of white cells in the i th row. We consider *shapes* of white rectangles. Each shape is a pair (p, q) that represents the dimensions of the rectangle. These shapes (and corresponding rectangles) are partially ordered by: $(p, q) < (p', q') \Leftrightarrow (p, q) \neq (p', q') \wedge p \leq p' \wedge q \leq q'$.

MAX WHITE RECTANGLES

Input: An array $Whites[1..m]$ that represents a staircase.

Output: Shapes of all maximal white rectangles in this staircase.

► **Lemma 21.** *MAX WHITE RECTANGLES problem can be solved in $\mathcal{O}(m)$ time.*

Proof. Assume that $Whites[0] = Whites[m+1] = -1$. Let us define two tables of size m :

$$NSVUp[i] = \max\{j : j < i, Whites[j] < Whites[i]\},$$

$$NSVDown[i] = \min\{j : j > i, Whites[j] < Whites[i]\}.$$

They can be computed in $\mathcal{O}(m)$ time by a folklore algorithm for the nearest smaller value table; see e.g. [13]. Then the problem can be solved as in Algorithm 1 presented below. After the first for-loop, for each maximal white rectangle R we have $MaxWidth[height(R)] = width(R)$, but we could have redundant values for non-maximal rectangles. In order to filter out non-maximal rectangles, we process the candidates by decreasing height and remove the ones that are dominated by the previous maximal rectangle in the partial order of shapes. ◀

■ **Algorithm 1** The first phase computes a set of shapes of type $(h, MaxWidth[h])$, at most one for each height h ; see also Figure 6. In the second phase only inclusion-maximal shapes from this set are reported.

ComputeCandidates:

$MaxWidth[1..m] := (0, \dots, 0)$

for $i := 1$ **to** m **do**

$h := NSVDown[i] - NSVUp[i] - 1$

$MaxWidth[h] := \max(MaxWidth[h], Whites[i])$

ReportMaximal:

$mw := 0$

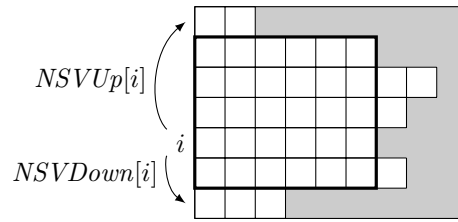
for $h := m$ **down to** 1 **do**

if $MaxWidth[h] > mw$ **then**

Report the shape $(h, MaxWidth[h])$

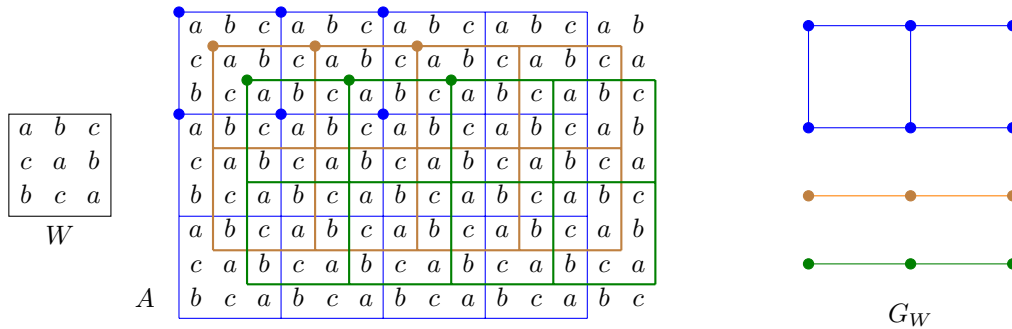
$mw := MaxWidth[h]$

► **Remark 22.** Note that the total area (and width) of a staircase can be large but the complexity of our algorithm is linear with respect to the number of rows, thanks to the small representation (array $Whites$).



■ **Figure 6** A maximal white rectangle containing row i is computed using the NSV tables for i .

Now our approach is graph-theoretic. The graph nodes correspond to occurrences of primitively rooted quartics. For a fixed primitively rooted quartic $W^{2,2}$ we consider the graph $G_W = (V, E)$, where V is the set of top-left corners of occurrences of $W^{2,2}$. Let $r = \text{height}(W)$ and $c = \text{width}(W)$. The edges in G connect vertex (i, j) with vertices $(i \pm r, j)$ and $(i, j \pm c)$, if they exist. See also Figure 7. This graph can be efficiently computed since we know its nodes due to Theorem 19.



■ **Figure 7** Graph G_W has 12 vertices that form two components with 3 vertices each (green and brown) and one component with 6 vertices (blue). Note the non-trivial occurrences of W in $W^{3,4}$.

► **Lemma 23.** *All graphs G_W , and their connected components, for all W which are primitive roots of quartics in A can be constructed in $\mathcal{O}(n^2 \log^2 n)$ time.*

Proof. We first compute all occurrences of primitively rooted quartics in A using Theorem 19. By Fact 3, there are $\mathcal{O}(n^2 \log^2 n)$ of them in total.

We can assign 2D-DBF identifiers (quadruples) to each of the occurrences and group the occurrences by distinct primitively rooted quartics via radix sort in $\mathcal{O}(n^2 \log^2 n)$ time. This gives us the vertices of G_W .

To compute the edges, we use an auxiliary $n \times n$ Boolean array D that will store top-left corners of occurrences of each subsequent primitively rooted quartic $W^{2,2}$.

Initially D is set to zeroes and after each W , all cells with ones are zeroed in $\mathcal{O}(|G_W|)$ time. Using this array and the positions of occurrences of $W^{2,2}$, the edges of G_W can be computed in $\mathcal{O}(|G_W|)$ time. It also allows to divide G_W into connected components via graph search in $\mathcal{O}(|G_W|)$ time. ◀

► **Theorem 24.** *All distinct quartics in an $n \times n$ array A can be computed in $\mathcal{O}(n^2 \log^2 n)$ time.*

32:14 The Number of Repetitions in 2D-Strings

Proof. We first apply Lemma 23. Now consider a fixed primitive W of height c and width r . Let us note that if $(i, j), (i', j')$ belong to the same connected component H of G_W , then $i \equiv i' \pmod{r}$ and $j \equiv j' \pmod{c}$. We say that a connected component H of G_W *generates* an occurrence of a power $W^{\alpha, \beta}$ if the $\alpha\beta$ occurrences of W that are implied by it belong to H . If $W^{\alpha, \beta}$ has an occurrence in A , then it is generated by some connected component H of G_W , unless $\min(\alpha, \beta) = 1$.

We say that $W^{\alpha, \beta}$ is a *maximal* power if there is no other power $W^{\alpha', \beta'}$ in A such that $\alpha' \geq \alpha$, $\beta' \geq \beta$, and $(\alpha', \beta') \neq (\alpha, \beta)$. Similarly, we consider powers that are maximal among ones that are generated by a connected component H . Let $MaxPowers_W(H)$ be the set of maximal powers generated by a connected component H . It can be computed in linear time using Lemma 21 as shown in Algorithm 2, which we now explain.

For each vertex (i, j) in H , we insert four points to a set S , which correspond to the four occurrences of W underlying the occurrence of quartic $W^{2,2}$ at position (i, j) . If S is treated as a set of white cells in a grid, then $W^{\alpha, \beta}$ for $\alpha > 1$ is a power generated by H if and only if the grid contains a white rectangle of shape (α, β) . For a cell $(i, j) \in S$, we denote $R[i, j] = \min\{p \geq 0 : (i, j + p) \notin S\}$. Assuming that the cells of S are sorted by non-increasing second component, each value $R[i, j]$ can be computed from $R[i, j + 1]$ in constant time, for a total of $\mathcal{O}(|S|)$ time. The sorting for all S can be done globally, using radix sort. Also, the array R can be stored globally and used for all S , cleared after each use. Finally, we process each maximal set of consecutive cells $(i, j), \dots, (i + m - 1, j) \in S$ that are located in the same column and apply Lemma 21 to solve the resulting instance of the MAX WHITE RECTANGLES problem. The total time required by this step is $\mathcal{O}(|S|)$.

■ **Algorithm 2** Computing $MaxPowers_W(H)$ for a component H of G_W .

```

S := ∅
foreach (i, j) in V(H) do
    a := ⌊i/r⌋; b = ⌊j/c⌋
    S := S ∪ {(a, b), (a + 1, b), (a, b + 1), (a + 1, b + 1)}
R[0..n, 0..n] := (0, ..., 0)
foreach (i, j) in S in non-increasing order of j do
    R[i, j] := R[i, j + 1] + 1
Result := ∅
foreach maximal set {(i, j), (i + 1, j), ..., (i + m - 1, j)} ⊆ S do
    Whites[1..m] := R[i..i + m - 1, j]
    Result := Result ∪ MAXWHITERECTANGLES(Whites)

remove redundant rectangles from Result
return Result

```

In the end we filter out the powers $W^{\alpha, \beta}$ that are not maximal in A similarly as in the proof of Lemma 21, using a global array $MaxWidth$. Let $W^{\alpha_1, \beta_1}, \dots, W^{\alpha_k, \beta_k}$ be the resulting sequence of maximal powers, sorted by increasing first component, and let $\alpha_0 = \beta_0 = 0$. Then the set of all quartics in A with primitive root W contains all $W^{2\alpha, 2\beta}$ over $\alpha_{p-1} < 2\alpha \leq \alpha_p$, $1 \leq 2\beta \leq \beta_p$, for $p \in [2, k]$. They can be reported in $\mathcal{O}(n^2 \log^2 n)$ total time over all W due to the upper bound of Theorem 16. ◀

6 Final Remarks

We showed that the numbers of distinct runs and quartics in an $n \times n$ array are $\mathcal{O}(n^2 \log^2 n)$. This improves upon previously known estimations. We also proposed $\mathcal{O}(n^2 \log^2 n)$ -time algorithms for computing all occurrences of primitively rooted quartics and all distinct quartics. A straightforward adaptation shows that for an $m \times n$ array these bounds and complexities all become $\mathcal{O}(mn \log m \log n)$.

We pose two conjectures for $n \times n$ 2D-strings:

- The number of 2D-runs is $\mathcal{O}(n^2)$.
- The number of distinct quartics is $\mathcal{O}(n^2)$.

References

- 1 Amihod Amir, Gary Benson, and Martin Farach. An alphabet independent approach to two-dimensional pattern matching. *SIAM Journal on Computing*, 23(2):313–323, 1994. doi:10.1137/S0097539792226321.
- 2 Amihod Amir, Itai Boneh, Panagiotis Charalampopoulos, and Eitan Konradovsky. Repetition detection in a dynamic string. In *27th Annual European Symposium on Algorithms, ESA 2019*, volume 144 of *LIPICs*, pages 5:1–5:18. Schloss Dagstuhl - Leibniz-Zentrum für Informatik, 2019. doi:10.4230/LIPICs.ESA.2019.5.
- 3 Amihod Amir, Ayelet Butman, Gad M. Landau, Shoshana Marcus, and Dina Sokol. Double string tandem repeats. In *31st Annual Symposium on Combinatorial Pattern Matching, CPM 2020*, volume 161 of *LIPICs*, pages 3:1–3:13. Schloss Dagstuhl - Leibniz-Zentrum für Informatik, 2020. doi:10.4230/LIPICs.CPM.2020.3.
- 4 Amihod Amir and Martin Farach. Efficient 2-dimensional approximate matching of non-rectangular figures. In *Proceedings of the Second Annual ACM/SIGACT-SIAM Symposium on Discrete Algorithms*, pages 212–223. ACM/SIAM, 1991. URL: <http://dl.acm.org/citation.cfm?id=127787.127829>.
- 5 Amihod Amir, Gad M. Landau, Shoshana Marcus, and Dina Sokol. Two-dimensional maximal repetitions. In *26th Annual European Symposium on Algorithms, ESA 2018*, volume 112 of *LIPICs*, pages 2:1–2:14. Schloss Dagstuhl - Leibniz-Zentrum für Informatik, 2018. doi:10.4230/LIPICs.ESA.2018.2.
- 6 Amihod Amir, Gad M. Landau, Shoshana Marcus, and Dina Sokol. Two-dimensional maximal repetitions. *Theoretical Computer Science*, 812:49–61, 2020. doi:10.1016/j.tcs.2019.07.006.
- 7 Alberto Apostolico and Valentin E. Brimkov. Fibonacci arrays and their two-dimensional repetitions. *Theoretical Computer Science*, 237(1-2):263–273, 2000. doi:10.1016/S0304-3975(98)00182-0.
- 8 Alberto Apostolico and Valentin E. Brimkov. Optimal discovery of repetitions in 2D. *Discrete Applied Mathematics*, 151(1-3):5–20, 2005. doi:10.1016/j.dam.2005.02.019.
- 9 Theodore P. Baker. A technique for extending rapid exact-match string matching to arrays of more than one dimension. *SIAM Journal on Computing*, 7(4):533–541, 1978. doi:10.1137/0207043.
- 10 Hideo Bannai, Tomohiro I, Shunsuke Inenaga, Yuto Nakashima, Masayuki Takeda, and Kazuya Tsuruta. The “runs” theorem. *SIAM Journal on Computing*, 46(5):1501–1514, 2017. doi:10.1137/15M1011032.
- 11 Hideo Bannai, Shunsuke Inenaga, and Dominik Köppl. Computing all distinct squares in linear time for integer alphabets. In *28th Annual Symposium on Combinatorial Pattern Matching, CPM 2017*, volume 78 of *LIPICs*, pages 22:1–22:18. Schloss Dagstuhl - Leibniz-Zentrum für Informatik, 2017. doi:10.4230/LIPICs.CPM.2017.22.
- 12 Jon Louis Bentley. Algorithms for Klee’s rectangle problems. Unpublished notes, Computer Science Department, Carnegie Mellon University, 1977.

- 13 Omer Berkman, Baruch Schieber, and Uzi Vishkin. Optimal doubly logarithmic parallel algorithms based on finding all nearest smaller values. *Journal of Algorithms*, 14(3):344–370, 1993. doi:10.1006/jagm.1993.1018.
- 14 Richard S. Bird. Two dimensional pattern matching. *Information Processing Letters*, 6(5):168–170, 1977. doi:10.1016/0020-0190(77)90017-5.
- 15 Panagiotis Charalampopoulos, Tomasz Kociumaka, Manal Mohamed, Jakub Radoszewski, Wojciech Rytter, Juliusz Straszyński, Tomasz Waleń, and Wiktor Zuba. Counting distinct patterns in internal dictionary matching. In *31st Annual Symposium on Combinatorial Pattern Matching, CPM 2020*, volume 161 of *LIPICs*, pages 8:1–8:15. Schloss Dagstuhl - Leibniz-Zentrum für Informatik, 2020. doi:10.4230/LIPICs.CPM.2020.8.
- 16 Panagiotis Charalampopoulos, Tomasz Kociumaka, Manal Mohamed, Jakub Radoszewski, Wojciech Rytter, and Tomasz Waleń. Internal dictionary matching. In *30th International Symposium on Algorithms and Computation, ISAAC 2019*, volume 149 of *LIPICs*, pages 22:1–22:17. Schloss Dagstuhl - Leibniz-Zentrum für Informatik, 2019. doi:10.4230/LIPICs.ISAAC.2019.22.
- 17 Maxime Crochemore. An optimal algorithm for computing the repetitions in a word. *Information Processing Letters*, 12(5):244–250, 1981. doi:10.1016/0020-0190(81)90024-7.
- 18 Maxime Crochemore, Christophe Hancart, and Thierry Lecroq. *Algorithms on strings*. Cambridge University Press, 2007.
- 19 Maxime Crochemore and Lucian Ilie. Analysis of maximal repetitions in strings. In *Mathematical Foundations of Computer Science 2007, 32nd International Symposium, MFCS 2007*, volume 4708 of *Lecture Notes in Computer Science*, pages 465–476. Springer, 2007. doi:10.1007/978-3-540-74456-6_42.
- 20 Maxime Crochemore and Lucian Ilie. Maximal repetitions in strings. *Journal of Computer and System Sciences*, 74(5):796–807, 2008. doi:10.1016/j.jcss.2007.09.003.
- 21 Maxime Crochemore, Lucian Ilie, and Liviu Tinta. The "runs" conjecture. *Theoretical Computer Science*, 412(27):2931–2941, 2011. doi:10.1016/j.tcs.2010.06.019.
- 22 Maxime Crochemore, Costas S. Iliopoulos, Marcin Kubica, Jakub Radoszewski, Wojciech Rytter, and Tomasz Waleń. Extracting powers and periods in a word from its runs structure. *Theoretical Computer Science*, 521:29–41, 2014. doi:10.1016/j.tcs.2013.11.018.
- 23 Maxime Crochemore and Robert Mercas. On the density of Lyndon roots in factors. *Theoretical Computer Science*, 656:234–240, 2016. doi:10.1016/j.tcs.2016.02.015.
- 24 Maxime Crochemore and Wojciech Rytter. Squares, cubes, and time-space efficient string searching. *Algorithmica*, 13(5):405–425, 1995. doi:10.1007/BF01190846.
- 25 Maxime Crochemore and Wojciech Rytter. *Jewels of stringology*. World Scientific, 2002. doi:10.1142/4838.
- 26 Antoine Deza, Frantisek Franek, and Adrien Thierry. How many double squares can a string contain? *Discrete Applied Mathematics*, 180:52–69, 2015. doi:10.1016/j.dam.2014.08.016.
- 27 Nathan J. Fine and Herbert S. Wilf. Uniqueness theorems for periodic functions. *Proceedings of the American Mathematical Society*, 16(1):109–114, 1965. doi:10.2307/2034009.
- 28 Johannes Fischer, Stepan Holub, Tomohiro I, and Moshe Lewenstein. Beyond the runs theorem. In *String Processing and Information Retrieval - 22nd International Symposium, SPIRE 2015*, volume 9309 of *Lecture Notes in Computer Science*, pages 277–286. Springer, 2015. doi:10.1007/978-3-319-23826-5_27.
- 29 Aviezri S. Fraenkel and Jamie Simpson. How many squares can a string contain? *Journal of Combinatorial Theory, Series A*, 82(1):112–120, 1998. doi:10.1006/jcta.1997.2843.
- 30 Frantisek Franek and Qian Yang. An asymptotic lower bound for the maximal number of runs in a string. *International Journal of Foundations of Computer Science*, 19(1):195–203, 2008. doi:10.1142/S0129054108005620.
- 31 Mathieu Giraud. Not so many runs in strings. In *Language and Automata Theory and Applications, Second International Conference, LATA 2008*, volume 5196 of *Lecture Notes in Computer Science*, pages 232–239. Springer, 2008. doi:10.1007/978-3-540-88282-4_22.

- 32 Amy Glen and Jamie Simpson. The total run length of a word. *Theoretical Computer Science*, 501:41–48, 2013. doi:10.1016/j.tcs.2013.06.004.
- 33 Dan Gusfield and Jens Stoye. Linear time algorithms for finding and representing all the tandem repeats in a string. *Journal of Computer and System Sciences*, 69(4):525–546, 2004. doi:10.1016/j.jcss.2004.03.004.
- 34 Lucian Ilie. A note on the number of squares in a word. *Theoretical Computer Science*, 380(3):373–376, 2007. doi:10.1016/j.tcs.2007.03.025.
- 35 Costas S. Iliopoulos, Dennis W. G. Moore, and William F. Smyth. A characterization of the squares in a Fibonacci string. *Theoretical Computer Science*, 172(1-2):281–291, 1997. doi:10.1016/S0304-3975(96)00141-7.
- 36 Dominik Kempa and Tomasz Kociumaka. String synchronizing sets: sublinear-time BWT construction and optimal LCE data structure. In *Proceedings of the 51st Annual ACM SIGACT Symposium on Theory of Computing, STOC 2019*, pages 756–767. ACM, 2019. doi:10.1145/3313276.3316368.
- 37 Tomasz Kociumaka, Jakub Radoszewski, Wojciech Rytter, and Tomasz Waleń. Internal pattern matching queries in a text and applications. In Piotr Indyk, editor, *Proceedings of the Twenty-Sixth Annual ACM-SIAM Symposium on Discrete Algorithms, SODA 2015*, pages 532–551. SIAM, 2015. doi:10.1137/1.9781611973730.36.
- 38 Roman M. Kolpakov and Gregory Kucherov. Finding maximal repetitions in a word in linear time. In *40th Annual Symposium on Foundations of Computer Science, FOCS 1999*, pages 596–604. IEEE Computer Society, 1999. doi:10.1109/SFFCS.1999.814634.
- 39 M. Lothaire. *Combinatorics on words, Second Edition*. Cambridge mathematical library. Cambridge University Press, 1997.
- 40 Shoshana Marcus and Dina Sokol. 2D Lyndon words and applications. *Algorithmica*, 77(1):116–133, 2017. doi:10.1007/s00453-015-0065-z.
- 41 Wataru Matsubara, Kazuhiko Kusano, Akira Ishino, Hideo Bannai, and Ayumi Shinohara. New lower bounds for the maximum number of runs in a string. In *Proceedings of the Prague Stringology Conference 2008*, pages 140–145, 2008. URL: <http://www.stringology.org/event/2008/p13.html>.
- 42 Simon J. Puglisi, Jamie Simpson, and William F. Smyth. How many runs can a string contain? *Theoretical Computer Science*, 401(1-3):165–171, 2008. doi:10.1016/j.tcs.2008.04.020.
- 43 Wojciech Rytter. The number of runs in a string: Improved analysis of the linear upper bound. In *23rd Annual Symposium on Theoretical Aspects of Computer Science, STACS 2006*, volume 3884 of *Lecture Notes in Computer Science*, pages 184–195. Springer, 2006. doi:10.1007/11672142_14.
- 44 Wojciech Rytter. The number of runs in a string. *Information and Computation*, 205(9):1459–1469, 2007. doi:10.1016/j.ic.2007.01.007.
- 45 Jamie Simpson. Modified Padovan words and the maximum number of runs in a word. *The Australasian Journal of Combinatorics*, 46:129–146, 2010. URL: http://ajc.maths.uq.edu.au/pdf/46/ajc_v46_p129.pdf.
- 46 Jens Stoye and Dan Gusfield. Simple and flexible detection of contiguous repeats using a suffix tree. *Theoretical Computer Science*, 270(1-2):843–856, 2002. doi:10.1016/S0304-3975(01)00121-9.
- 47 Esko Ukkonen. On-line construction of suffix trees. *Algorithmica*, 14(3):249–260, 1995. doi:10.1007/BF01206331.

A Alternative Algorithm for the Proof of Lemma 21

An alternative, space efficient and more direct algorithm that does not use additional tables *NSVDown* and *NSVUp*, is shown below. The algorithm computes only the table *MaxWidth*. Then, we can use the second phase from Algorithm 1. We assume that the table *MaxWidth* is initially filled with zeros.

32:18 The Number of Repetitions in 2D-Strings

■ **Algorithm 3** Alternative implementation of the first phase in Algorithm 1.

```
Whites[0] := Whites[m + 1] := 0
S := empty stack; push(S, 0)
for  $i := m$  down to 0 do
  while  $Whites[i] < Whites[top(S)]$  do
     $k := top(S)$ ;  $h := top(S) - i - 1$ 
     $MaxWidth[h] := \max(MaxWidth[h], Whites[k])$ 
    pop(S)
  if  $Whites[top(S)] = Whites[i]$  then pop(S)
  push(S, i)
```

The algorithm is a version of a folklore algorithm for the Nearest Smaller Values problem and correctness can be shown using the same arguments. If $Whites[i] < Whites[i + 1]$, then the algorithm produces shapes of all Max White Rectangles anchored at $i + 1$, otherwise $i + 1$ is “nonproductive”. Observe that $i + 1 = top(S)$ when we start processing $i \geq 1$.

Let us analyze the time complexity of the algorithm. In total $m + 2$ elements are pushed to the stack. Each iteration of the while-loop pops an element, so the total number of iterations of this loop is $\mathcal{O}(m)$. Consequently, the algorithm works in $\mathcal{O}(m)$ time. In the end one needs to filter out non-maximal rectangles as in the previous proof of Lemma 21.