Rectangular Tile Covers of 2D-Strings

Jakub Radoszewski
University of Warsaw, Poland

Wojciech Rytter
University of Warsaw, Poland

Juliusz Straszyński
University of Warsaw, Poland

Tomasz Waleń
University of Warsaw, Poland

Wiktor Zuba
University of Warsaw, Poland
CWI, Amsterdam, The Netherlands

Abstract

We consider tile covers of 2D-strings which are a generalization of periodicity of 1D-strings. We say that a 2D-string A is a *tile cover* of a 2D-string S if S can be decomposed into non-overlapping 2D-strings, each of them equal to A or to A^T, where A^T is the transpose of A. We show that all tile covers of a 2D-string of size N can be computed in $\mathcal{O}(N^{1+\varepsilon})$ time for any $\varepsilon > 0$. We also show a linear-time algorithm for computing all 1D-strings being tile covers of a 2D-string.

2012 ACM Subject Classification Theory of computation → Pattern matching

Keywords and phrases tile cover, periodicity, efficient algorithm

Digital Object Identifier 10.4230/LIPIcs.CPM.2022.23

Funding *Jakub Radoszewski*: Supported by the Polish National Science Center, grant no. 2018/31/D/ST6/03991.
Juliusz Straszyński: 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 Netherlands Organisation for Scientific Research (NWO) through Gravitation-grant NETWORKS-024.002.003.

Acknowledgements We thank Panagiotis Charalampopoulos for helpful discussions.

1 Introduction

A 1D-string (or simply a string) is a finite sequence of letters. A 2D-string S is a rectangular 2D-matrix consisting of n rows being strings of equal length m. The dimensions of the 2D-string are $n \times m$ and its size is $|S| = N = n \cdot m$. We say that a 1D-string S has period p if $S[i] = S[i+p]$ for all $i = 1, \dots, |S| - p$. We say that a 2D-string S has horizontal (vertical) period p if each row (column, respectively) of S has period p.

Periodicity in 2D-strings has different properties than in 1D-strings. Let us consider an example based on the following known notions of periodicity. A *run* in a 1D-string (2D-string, respectively) S is a maximal periodic factor of S (maximal submatrix that is periodic in both dimensions). Two natural 2D-generalizations of squares in a 1D-string are known; a *quartic* is a configuration that is composed of 2×2 occurrences of an array W and a *tandem* is a configuration consisting of two occurrences of an array W that share one side. A string of length n has $\mathcal{O}(n)$ distinct square factors [8, 7, 14] and $\mathcal{O}(n)$ runs [11, 3]. However, it was recently shown that a 2D-string of size N can have $\Omega(N^{3/2})$ distinct tandems, $\Omega(N \log N)$ distinct quartics and $\Omega(N \log N)$ runs [9].

Motivated by these differences, we introduce a natural generalization of periodicity to 2D-strings which we call tile covers. A 1D-string S has a full period P if $S = P^k$ for some positive integer k. We say that a 2D-string A is a *2D-tile cover* or simply *tile cover* of an $n \times m$ 2D-string S if S can be decomposed into (non-overlapping) 2D-strings each of them equal to A or to A^T, where A^T is the transpose of A (the i-th row of A becomes in A^T the i-th column); see Figure 1. In this paper we consider the following problem.

Tile covers problem. Compute efficiently all tile covers A of a given 2D-string S.

We consider this problem in full generality as well as a special case in which A is a 1D-string (a rectangle consisting of a single row) which we call the **1D-tile covers problem**. Note that for a 2D-string of size N, there are at most N tile covers; each of them corresponds to a submatrix of S starting in its top left corner. In particular, each tile cover of S can be represented in $\mathcal{O}(1)$ space.

▶ **Observation 1.** *In case when S is a 1D-string, the tile cover problem is trivial: a 1D-string A is a tile cover of a 1D-string S if and only if S is a string power of A. However in case of 2D-strings the problem becomes complicated, even for 1D-tile covers.*

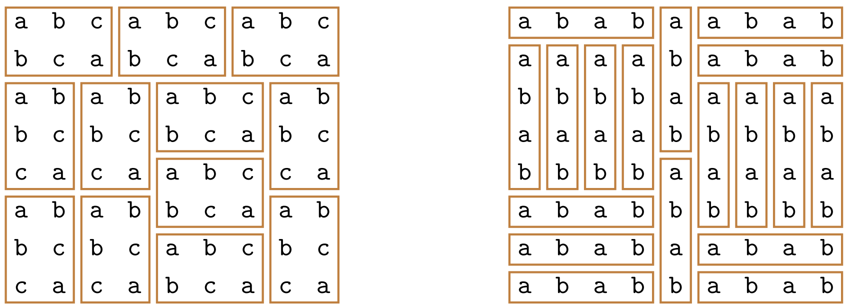

Figure 1 Two examples of tile coverings of 2D-strings; the first one is by a 2×3 2D-tile cover and the second one is by a 1D-tile cover of length 4. Let us notice that in the case of 1D-tile covers both `abab` and its primitive root `ab` are 1D-tile covers of the 2D-string.

A known generalization of periodicity in 1D-strings is quasiperiodicity. A string C is a cover of a 1D-string S if S can be created by possibly overlapping occurrences of C; see e.g. [2, 4]. Versions of covers of 2D-strings were studied before; the main difference between these notions and tile covers is that tile covers do not allow covering the text with overlapping occurrences of the cover. In [5] two notions of covers of 2D-strings, called 1D-covers and 2D-covers, were considered. A 2D-string C is a *2D-cover* of a 2D-string T if each position of T is inside an occurrence of C in T. Various algorithms computing 2D-covers were presented in [5, 6, 13]. A (1D) string C is a *1D-cover* of a 2D-string T if each position of T is inside an occurrence of C or C^T. A linear-time algorithm computing all 1D-covers of a string was proposed in [5].

Our Results. Let S be a 2D-string of size N. We show that:

- all 1D-tile covers of S can be computed in $\mathcal{O}(N)$ time;
- all tile covers of S can be computed in $\mathcal{O}(N^{1+\varepsilon})$ time for any $\varepsilon > 0$.

Let us recall that each tile cover can be represented in $\mathcal{O}(1)$ space as a submatrix of S.

2 1D-Tile covers of 2D-strings

In this section we show how to compute 1D-tile covers of an $n \times m$ 2D-string S of size N. Henceforth by ℓ we denote the length of the 1D-tile cover.

Let us recall some notation and properties of periodicity of 1D-strings. A string B that is both a prefix and a suffix of a string U is called a border of U. A factor F of a string U is called proper if $|F| < |U|$. By $root(U)$, called the primitive root of U, we denote the shortest string Z such that U is a power of Z. We say that U is primitive if $root(U) = U$. The string $root(U)$ is primitive. Let us also recall the solution to the following classical word equation; cf. [12].

▶ **Lemma 2.** *If strings X, Y satisfy $XY = YX$, then there exists a string Z such that $X = Z^x$ and $Y = Z^y$ for some positive integers x, y.*

2.1 Unary 1D-tile covers

The unary case is simpler, but not completely trivial.

▶ Remark 3. The number of distinct unary 1D-tile coverings is potentially exponential, even if $\ell = 2$. For example, there are 12,988,816 ways to tile a standard 8×8 chessboard with dominoes (to tile the 8×8 unary 2D-text by a linear unary tile of length $\ell = 2$). The numbers of distinct domino tilings of a $2 \times n$ text are Fibonacci numbers.

We define the following auxiliary $n \times m$ table D_ℓ: the first row is a prefix of $(1, 2, \ldots, \ell)^\infty$, and each subsequent row results by adding 1 to the elements of the previous row, substituting each $\ell + 1$ with 1; see Figure 2.

1	2	3	4	5	6	1	2	3
2	3	4	5	6	1	2	3	4
3	4	5	6	1	2	3	4	5
4	5	6	1	2	3	4	5	6
5	6	1	2	3	4	5	6	1
6	1	2	3	4	5	6	1	2
1	2	3	4	5	6	1	2	3
2	3	4	5	6	1	2	3	4

Figure 2 Illustration of the proof of Lemma 5 for $n = 8$, $m = 9$, $\ell = 6$. The two framed rectangles are balanced; however the remaining 2×3 rectangle A is not, hence a unary 8×9 matrix does not have a 1D-tile cover of length 6 (even though 6 divides $72 = 8 \cdot 9$).

We say that a submatrix of D_ℓ is *balanced* if each integer in $\{1, \ldots, \ell\}$ occurs the same number of times in this submatrix.

▶ **Observation 4.** *For $0 < k, r < \ell$, each $k \times r$ submatrix A of D_ℓ is* not *balanced.*

Proof. The proof is by contradiction. Assume each of the numbers $1, \ldots, \ell$ occurs in A the same number of times. The integer at position $\min(k, r)$ in the first row of A occurs in the first $\min(k, r)$ rows of A. Hence each integer in A should occur at least $\min(k, r)$ times, altogether we have at least $\min(k, r) \cdot \ell$ occurrences in A. However, it is impossible since A has only $k \cdot r$ positions and $k \cdot r < \min(k, r) \cdot \ell$ due to the inequality $k, r < \ell$. ◀

▶ **Lemma 5.** *A unary 2D-string of size $n \times m$ has a (unary) 1D-tile cover of length ℓ if and only if ℓ divides n or m.*

Proof. If ℓ divides m, then the text can be covered trivially with the use of horizontal occurrences; symmetrically with the use of vertical ones if ℓ divides n. Let us consider the other implication.

Each occurrence of a 1D-tile cover of length ℓ is balanced in the array D_ℓ, hence the whole 2D-string can be covered only if D_ℓ is balanced. Since every submatrix of D_ℓ of dimensions $k \times \ell$ and $\ell \times k$ is balanced, the problem reduces to the case of a matrix of dimensions $m \bmod \ell$ and $n \bmod \ell$, both smaller than ℓ, which is covered by Observation 4. ◀

▶ Remark 6. Let us note that it is not sufficient in the lemma for ℓ to divide $n \cdot m$; see Figure 2.

2.2 Combinatorics of non-unary 1D-tile covers

In this section we consider only non-unary texts S and non-unary tile covers (it is impossible for a non-unary text to have a unary tile cover or for a unary text to have a non-unary tile cover). Assume that the first row of S starts with the letter a. Consequently each 1D-tile cover starts with an occurrence of the letter a.

▶ **Definition 7.** *For a nonempty 2D-string U we define*

$$ratio(U) \;=\; \tfrac{\#_a(U)}{|U|}$$

where $\#_c(U)$ is the number of occurrences of the letter c in U.

We say that two 2D-strings are *similar*, written $U_1 \sim U_2$, if $ratio(U_1) = ratio(U_2)$. Obviously if U is a tile cover of S, then $U \sim S$.

Notice that since the top left position of S has to be covered, any 1D-tile cover U has to be a prefix of the first row or the first column of S, and by symmetry also the suffix of the last row or the last column of S. Henceforth we consider the case in which U is a prefix of the first row of S and a suffix of the last row of S; the remaining cases can be handled in a symmetric way.

▶ **Definition 8.** *In this section we say that a 1D-string U of length at most* $\min(n, m)$ *is a* candidate *if it satisfies the following conditions:*
- *U is a prefix of the first row of S and a suffix of the last row of S;*
- *$U \sim S$;*
- *The first row of S belongs to $(U \cup a)^*$, that is, it can be factorized into factors equal to U and a.*

▶ **Observation 9.** *If U is a 1D-tile cover of S, then U is a candidate.*

▶ **Observation 10.** *If U is a 1D-tile cover of S, then so is $root(U)$.*

The following auxiliary lemma plays an important role in a proof of Lemma 12 that gives a key characterization of candidates.

▶ **Lemma 11.** *Assume $U = L\,a^k\,R = R\,a^k\,L$, U is primitive and non-unary, and $L, R \neq \varepsilon$. Then $ratio(a^k\,R) > ratio(U)$.*

Proof. We have that $k > 0$, as otherwise $U = LR = RL$, which implies that U is not primitive (see Lemma 2). The word equation from the statement of the lemma is equivalent to $a^k\,U = (a^k\,L)(a^k\,R) = (a^k\,R)(a^k\,L)$, hence, again by Lemma 2, strings $a^k\,U$ and $a^k\,R$ have a common primitive root, which concludes that $ratio(a^k\,U) = ratio(a^k\,R)$. However, since U is non-unary, we have that $ratio(a^k\,R) = ratio(a^k\,U) > ratio(U)$. ◀

▶ **Lemma 12.** *Take two candidates U, V, such that U is primitive and $|U| < |V|$. Then V is a power of U.*

Proof. Both U and V are candidates, which implies that U is a border of V. At the same time, since U is a candidate, the occurrence of V that covers the top left corner of S can be factorized into occurrences of U, letters a and a (possibly empty) string R – a proper prefix of U – at its end; see Figure 3. Let $\mathcal{F}$ denote this factorization. We will show that $R = \varepsilon$, which will conclude that V is a power of U. Indeed, otherwise we would have $ratio(V) > ratio(U)$, contradicting the definition of a candidate.

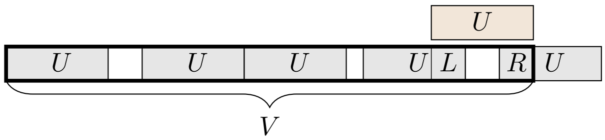

Figure 3 The series of grey boxes in the main line represents a factorization of the first row of 2D-string S into (non-unary) U's separated by a^*'s (white spaces, possibly zero a's). The upper brown box corresponds to the suffix of V equal to U. We focus on the first occurrence of V in the text. If additionally we assume that $ratio(U) = ratio(V)$, and that U is primitive, then V must be a power of U (that is, there are no white spaces, and the last U ends at the same point as V).

The length-$|U|$ suffix of V, by the factorization $\mathcal{F}$, has a form La^kR for some (possibly empty) suffix L of U and $k \geq 0$. However, this suffix is equal to U, so $U = La^kR$. At the same time, since R is a prefix of U, we also have that $U = RWL$ for some string W. From these two decompositions of U we see that $ratio(a^k) = ratio(W)$, i.e. $W = a^k$.

This will allow us to apply Lemma 11. First, if any of the strings L, R is empty and $k > 0$, by Lemma 2, we obtain that U is unary, and so is V, which concludes the proof. If any of L, R is empty and $k = 0$, then $R = \varepsilon$ (since R was chosen as a proper prefix of U) and we obtain the conclusion as shown before. Otherwise we can indeed apply Lemma 11 and obtain that $ratio(a^k R) > ratio(U)$. From the aforementioned factorization $\mathcal{F}$ of V we obtain $ratio(V) > ratio(U)$, which contradicts the definition of a candidate. ◀

▶ **Corollary 13.** *Assume a non-unary text S has a 1D-tile cover. Then the shortest 1D-tile cover of S is the shortest candidate U. It is a primitive string. All other 1D-tile covers are powers of U, though not all powers of U are necessarily 1D-tile covers of S.*

The corollary suggests the following algorithm for finding the shortest 1D-tile cover:

1. Find the shortest primitive candidate U.
2. Then check if it is a 1D-tile cover.

The first step is easy because it involves only 1D-strings: first row/column and the last row/column. The second step can be done using a greedy approach.

However, testing which powers of U are 1D-tile covers requires a slightly different number-theoretic approach.

2.3 Computing non-unary candidates

All 1D-tile covers U satisfying $|U| > \min(n, m)$ can be trivially computed, hence we later assume that the length of the cover is at most $\min(n, m)$.

We use the following algorithm to compute all candidates.

Algorithm 1 ALL-CAND(S).

Compute *ratio*(S)
Cand is initially the set of all proper borders of $S_1\$S_2$ of length $\leq n$, where S_1 and S_2 are respectively the first and the last row of S and \$ is a special symbol
using prefix-sums computation we compute *ratio*(Z) for each prefix of S_1
remove from *Cand* all U such that *ratio*$(U) \neq$ *ratio*(S)
foreach $U \in$ *Cand* **do**
    if $S_1 \notin (U \cup a)^*$ **then** remove U from *Cand*
    // It is checked in $\mathcal{O}(m)$ time per each U
return *Cand*

▶ **Corollary 14.** *ALL-CAND(S) computes in $\mathcal{O}(N)$ time all candidates.*

Proof. Borders of a string of length m can be computed in $\mathcal{O}(m)$ time [10]. We have $|Cand| \leq \min(m, n)$ and each $U \in Cand$ is checked in $\mathcal{O}(m)$ time using linear-time pattern matching [10]. ◀

2.4 Testing a single non-unary candidate

Assume # does not occur in S. In the course of the next algorithm certain entries will be marked, i.e., changed to #. For a 2D-string U we define two operations:

- Match(i, j, U): return true if and only if there is a full occurrence of U starting in position (i, j) in S,
- Mark(i, j, U): changes each symbol in S in the occurrence of U starting in (i, j) to #.

▶ Remark 15. The following algorithm GREEDY is also well defined for a 2D-string U, but it does not work correctly for all tile cover candidates which are not 1D-strings. We reuse it later in Section 3.2.1.

Algorithm 2 GREEDY(S, U).

Output: True if a non-unary 2D-string U is a 1D-tile cover of S
for $i := 1$ **to** n **do**
    for $j := 1$ **to** m **do**
        if $S[i, j] \neq \#$ **then**
            if Match(i, j, U) **then**
                ▷ *choosing occurrence of* U
                Mark(i, j, U)
            else if Match(i, j, U^T) **then**
                ▷ *choosing occurrence of* U^T
                Mark(i, j, U^T)
            else
                return false
▷ *All positions are now marked*
return true

▶ **Theorem 16.** *Assume U is a 1D-string. Then GREEDY(S, U) checks if U is a 1D-tile cover of S in $\mathcal{O}(N)$ time. Covering of S with U is unique (occurrences of U forming the tile cover can be chosen only in a single, unique way).*

Proof.

Correctness. When we are processing the i-th row then the preceding rows are already completely marked. Hence each non-marked position in this row should be marked now by an occurrence of U or U^T starting in this row. If $\mathsf{Match}(i, j, U)$, then U must be used instead of U^T.

Indeed, U is non-unary. Let k be its first position containing letter different from a. $\mathsf{Match}(i, j, U)$ determines an existence of an uncovered letter different from a at position $(i, j + k - 1)$. This position cannot be covered with an occurrence of U beginning further (k is the first such position), nor with a U^T, which determines that U must be used at position (i, j).

Complexity. Each operation Match can be done in $\mathcal{O}(|U|)$ time. It is amortized by marking $|U|$ positions which were not marked previously. Consequently, the total time is $\mathcal{O}(N)$. ◀

2.5 Finding all 1D-tile covers in non-unary texts

Denote by $\gcd(M)$ the greatest common divisor of integers in a set M. We use the following algorithm (see also Figure 4).

Algorithm 3 ALL-TILES(S).

Output: All lengths of 1D-tile covers
U := shortest element of $Cand(S)$
if GREEDY(S, U)=false then return $\emptyset$
now GREEDY(S, U) can partition S into horizontal occurrences of U and vertical occurrences of U^T
we merge consecutive horizontal occurrences and consecutive vertical occurrences in possibly larger disjoint horizontal and vertical strips
M := set of lengths of obtained strips
$d := \gcd(M)$
return all powers of U whose lengths divide d

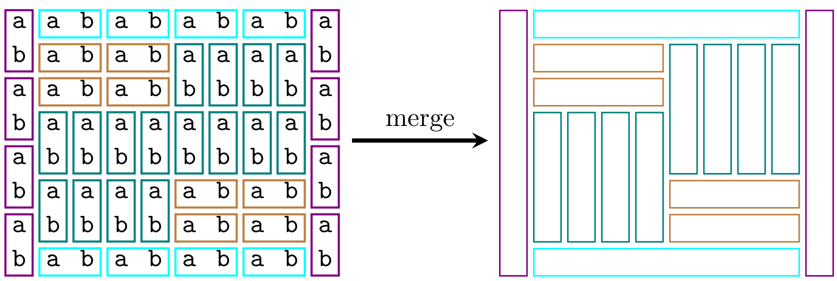

Figure 4 Merging of occurrences of the smallest 1D-tile cover in the covering to compute the larger ones. Here rectangles of length 4 and 8 appear, their greatest common divisor is 4, hence 1D-tile covers have lengths 2 and 4 (multiples of 2 and divisors of 4).

▶ **Theorem 17.** *All 1D-tile covers of a 2D-string of size N can be computed in $\mathcal{O}(N)$ time.*

Proof.

Correctness. Let U be the shortest 1D-tile cover. By Corollary 13 only the powers of U can be 1D-tile covers. The merged rectangles partition the 2D-string into 1D-parts. If the length of a given power of U divides the length of each rectangle, then each part of the division is trivially covered, hence the power is a 1D-tile cover. On the other hand if a given power of U is a 1D-tile cover, then by applying Algorithm 3 with it as the candidate we would obtain a different set of rectangles. This however would contradict the uniqueness of covering of S with U given by Theorem 16.

Complexity. We find the shortest 1D-tile cover with the use of Algorithms 1 and 2 both running in $\mathcal{O}(N)$ time. Algorithm 2 as a byproduct returns the set of $N/|U|$ rectangles, which can be merged in $\mathcal{O}(N/|U|)$ time. The greatest common divisor of their lengths is computed in exactly the same time. ◀

3 2D-Tile covers in 2D-strings

In this section we consider tile covers of shape $d \times \ell$, where $d \leq \ell$ (we find the other ones as tile covers of S^T).

3.1 Unary 2D-tile covers

▶ **Lemma 18.** *Unary $d \times \ell$ 2D-string is a tile cover of a unary $n \times m$ 2D-string if either case applies:*
(a) *d is a divisor of one dimension, and ℓ is a divisor of the other dimension of S,*
(b) *both d and ℓ divide the same dimension of S and the length of the other dimension is of the form $a \cdot d + b \cdot \ell$, for integers $a, b \geq 0$.*

Proof. From Lemma 5 we know that both d and ℓ have to divide one of the dimensions n or m (unary tiling with $d \times \ell$ rectangle easily divides into a one with $1 \times d$ or $1 \times \ell$ rectangles). If each dimension of U divides a different dimension of S (case (a)), then we can tile cover S with U trivially with only occurrences of U or occurrences of U^T. For the other case we assume, that both d and ℓ divide m.

If $n = a \cdot d + b \cdot \ell$ for integers $a, b \geq 0$ we can divide S into two parts of size $(a \cdot d) \times m$ and $(b \cdot \ell) \times m$, and cover the first one with only U's and the second one with only U^T's.

On the other hand if U is a tile cover of S, then the tiling divides the first column into segments of lengths d or ℓ, hence n must be of a form $a \cdot d + b \cdot \ell$ for some integers $a, b \geq 0$. ◀

3.2 Non-unary 2D-tile covers – testing a candidate

We make use of 2-dimensional properties of 2D-tiles related to symmetry and horizontal periodicity. We assume later in this section that all considered 2D-tiles are of shape $d \times \ell$, where $d \leq \ell$.

▶ **Definition 19.** *A square matrix A is called symmetric if $A = A^T$. A matrix is called horizontally periodic (H-periodic, in short) if its i-th column equals its $(i + d)$-th column, for $i \leq \ell - d$.*

Denote by $Pref(U)/Suf(U)$ the square matrix consisting of the first/last d columns of U.

▶ **Definition 20.** *Define*

$\gamma(U) \equiv Pref(U),\ Suf(U)$ *are symmetric and* U *is H-periodic.*

3.2.1 Case: not $\gamma(U)$

Observe that the algorithm GREEDY(S, U) can be applied also to 2D-tiles. The main deterministic choices of this algorithm are whether to use U or U^T. In this algorithm the priority is given to U. It works correctly for 1D-tiles, unfortunately such simple solution is incorrect for 2D-tile covers U, see Figure 5.

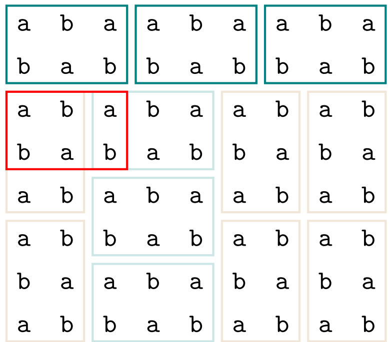

Figure 5 The algorithm GREEDY(S, U) from Section 2 does not work in this case: we start in the top-left corner and take as U the prefix of S of shape 2×3 (2 rows, 3 columns) and fill the first two rows this way. However when we do the same thing when we start covering the third row we cannot continue later and GREEDY returns *false*. It is incorrect because U covers S in a different non-greedy way.

▶ **Lemma 21.** *If $Pref(U)$ is not symmetric, or U is not H-periodic then GREEDY(S, U) works correctly.*

Proof. The only way, the algorithm GREEDY can give a bad answer is the case, where both $\mathsf{Match}(i, j, U)$ and $\mathsf{Match}(i, j, U^T)$ return true in a given position (such that all of the previous positions are covered), and choose to use $\mathsf{Mark}(i, j, U)$ even though, the right occurrence to choose is U^T.

In the case, where $Pref(U)$ is not symmetric U and U^T cannot both match in any position ($U[1..d, 1..d] \neq U^T[1..d, 1..d]$).

If U is not H-periodic, then if $\mathsf{Match}(i, j, U)$ return true, then one of the $d \times d$ submatrices of S with its top-left corner at position (i, k), where $j < k < j + l, k \bmod d = j \bmod d$ is different from $Pref(U)$.

If $\mathsf{Mark}(i, j, U^T)$ was used, then position (i, k) cannot be covered neither with $\mathsf{Mark}(i, p, U)$ for $j < p \leq k, p \bmod d = j \bmod d$ nor with $\mathsf{Mark}(i, k, U^T)$, and only such occurrences are available to use. ◀

Denote by $\hat{U}$ the matrix resulting from U by reversing each row, and then reversing each column.

▶ **Lemma 22.** *If $Suf(U)$ is not symmetric then GREEDY$(\hat{S}, \hat{U})$ returns true iff U is a tile cover of S since it is never possible to match both U and U^T in the same position.*

The last two lemmas give simple linear time tile test for the case when $Pref(U)$ or $Suf(U)$ is not symmetric or U is not H-periodic.

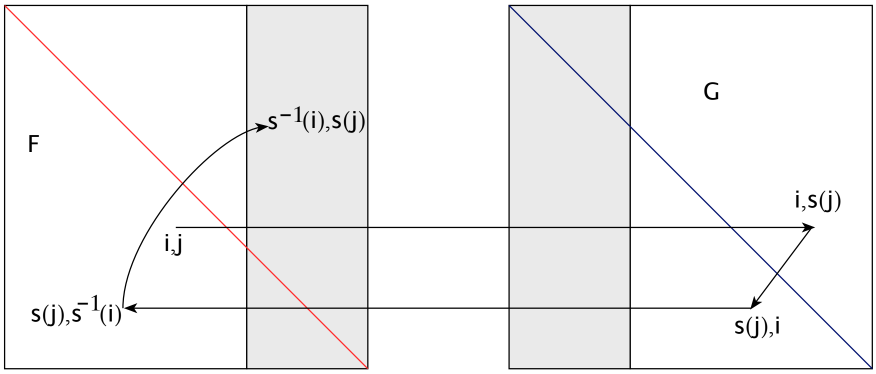

Figure 6 Migration of an element in the matrix F via the matrix G.

3.2.2 Reduction to case $\gcd(d, \ell) = 1$

It may be the case, that both S and U are in fact composed of smaller, symmetric $z \times z$ matrices. If all of those submatrices are equal, then the tile covering of S may not be unique even though it is not unary. In this section we show, that it is only possible for $z = \gcd(d, \ell)$, and that the problem can be reduced to the case, where $\gcd(d, \ell) = 1$. This also simplifies the algorithm in the next section.

Assume operations $\oplus$, $\ominus$ are addition and subtraction modulo k, respectively.

▶ **Fact 23.** *Assume* $\gcd(r, k) = 1$. *If* $Z \subseteq \{0, 1, 2, \ldots, k-1\}$ *is non-empty and has a property, that if* $x \in Z$ *implies* $x \oplus r \in Z$*, then* $Z = \{0, 1, 2, \ldots, k-1\}$.

In the lemma below we count rows and columns starting from zero.

▶ **Lemma 24.** *Assume* $0 < r < k$ *and* $\gcd(k, r) = 1$. *Let matrices* A, B *be of shapes* $k \times r$, $k \times (k - r)$*, respectively, and* $F = A \cdot B$, $G = B \cdot A$ *(where* $\cdot$ *denotes horizontal concatenation of matrices).*

If F, G *are symmetric, then all elements* $F[i \oplus x,\ j \ominus x]$ *are equal, for given* i, j *and all* $x \in \{0, 1, 2, \ldots, k-1\}$.

Proof. Let $s(x) = x \ominus r$. Then $F[i, j] = G[i, s(j)]$; see Figure 6.

Due to symmetry of G we have $G[i, s(j)] = G[s(j), i]$. Then $G[s(j), i] = F[s(j), s^{-1}(i)]$, and using symmetry of F we have

$$F[s(j), s^{-1}(i)] = F[s^{-1}(i), s(j)] = F[i \oplus r, j \ominus r]$$

Consequently $F[i, j] = F[i \oplus r, j \ominus r]$. The thesis follows now from Fact 23, by iterating the last equality. ◀

Denote $z = \gcd(d, \ell)$ and $k = d/z$.

▶ **Lemma 25.** *Assume* $\gamma(U)$. *Let us decompose* U *into disjoint submatrices* $z \times z$. *Then each of these submatrices is symmetric.*

Proof. Let us treat each of these $z \times z$ submatrices as single elements. Then we obtain the $k \times (\ell/z)$ matrix U'. We can apply Lemma 24 to $F = Pref(U')$, $G = Suf(U')$. Then Lemma 24 implies that each of our $z \times z$ sub-matrices equals a $z \times z$ sub-matrix on the main diagonal of $Pref(U)$ (or $Suf(U)$), consequently it is symmetric due to symmetry of diagonal $z \times z$ submatrices (see Figure 7). ◀

a	b	d	e	g	h	a	b	d	e
b	c	e	f	h	i	b	c	e	f
d	e	g	h	a	b	d	e	g	h
e	f	h	i	b	c	e	f	h	i
g	h	a	b	d	e	g	h	a	b
h	i	b	c	e	f	h	i	b	c

Figure 7 The figure illustrate the most general example of U of size 6×10 such that $\gamma(U)$. Notice that it is composed of symmetric submatrices 2×2 ($z = 2 = \gcd(6, 10)$) of only $k = 3 = 6/2$ types.

▶ **Corollary 26.** *If $\gamma(U)$ and $\gcd(d, \ell) > 1$ then we can reduce in linear time our problem to the case of $d' \times \ell'$ candidate U', where $\gamma(U')$ and $\gcd(d', \ell') = 1$.*

Proof. We decompose U and S into disjoint $z \times z$ submatrices. By Lemma 25 we know that the submatrices composing U are symmetric. If a decomposition of S contains a non-symetric matrix, then U cannot be a tile cover of S. Otherwise we replace each $z \times z$ submatrix with 1×1 submatrix, with symbol identifying the corresponding submatrix. Then the resulting 2D-texts U', S' prove the thesis. ◀

3.2.3 Case: $\gamma(U)$ and $\gcd(d, \ell) = 1$

The following fact can be shown using Fact 23 and similar arguments as in the proof of Lemma 24.

▶ **Fact 27.** *Assume $\gamma(U)$ and $\gcd(d, \ell) = 1$. Then if two distinct columns of $Pref(U)$ are equal, then U is unary.*

Denote by $first(U)$ the first column of U, by $first(Suf(U))$ the first column of $Suf(U)$ and by $Col(i, j, S)$ the fragment of size d of the j-th column of S starting in row i. We use the following algorithm.

Algorithm 4 GREEDY'(S, U).

Output: True if U is a 2D-tile cover
▷ *Assume U is non-unary*
for $i := 1$ **to** n **do**
    for $j := 1$ **to** m **do**
        if $S[i, j] \neq \#$ **then**
            if Match(i, j, U) **and** $First(Suf(U)) \neq Col(i, j + \ell, S)$ **then**
                ▷ *choose occurrence of U*
                Mark(i, j, U)
            else if Match(i, j, U^T) **then**
                ▷ *choose occurrence of U^T*
                Mark(i, j, U^T)
            else
                return false
▷ *All positions are now marked*
return true

▶ **Example 28.** Figure 8 shows how our algorithm is working. The figure illustrates the case when $First(Suf(U)) = Col(i, j+\ell, S)$ for $i = j = 1$, (the first column of $Suf(U)$ starts in the 4-th column of S). Hence instead of using $\mathsf{Mark}(1, 1, U)$ we execute $\mathsf{Mark}(1, 1, U^T)$.

The next (i, j) with $S[i, j] \neq \#$ is $(1, 3)$. For $j = 3$ we have $j+\ell = 6$ and $First(Suf(U)) \neq Col(1, 6, S)$, hence we perform here $\mathsf{Mark}(1, 3, U)$.

After using $\mathsf{Mark}(5, 1, U^T)$ the next (i, j) with $S[i, j] \neq \#$ is $(3, 3)$. Here $S[i, j+3] = \#$, hence we perform $\mathsf{Mark}(3, 3, U)$.

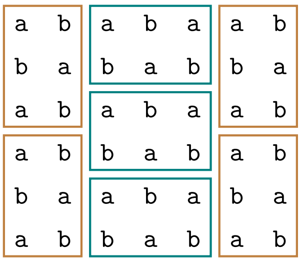

Figure 8 U is here the 2×3 prefix of S, $d = 2$, $\ell = 3$. The algorithm GREEDY' starts with U^T (due to additional comparison of two columns), while a naive greedy would start with U (and later fail).

▶ **Theorem 29.** *We can test if a given 2D-tile candidate is a 2D-tile cover in $\mathcal{O}(N)$ time.*

Proof. If not $\gamma(U)$, then we use Algorithm 2. This case was already covered by Lemmas 21 and 22. Otherwise we use Algorithm 4. Assume, that $\gamma(U)$ and $\gcd(d, \ell) = 1$.

If $\mathsf{Match}(i, j, U)$, then $First(Suf(U)) \neq Col(i, j+\ell, S)$ represents a break in H-periodicity (possibly due to an occurrence of # or the end of the text). In this case if the algorithm decides to use U^T instead of U at the next not covered positions $(i, j+d), (i, j+2d), \ldots, (i, j+kd)$ for $\ell - d < kd < \ell$, the use of U will not be possible, and hence it will have to use U^T.

However then, when trying to cover position $(i, j+kd)$ for $\ell - d < kd < \ell$ it will be also unable to use U^T due to the same break of the period. If however $First(Suf(U)) = Col(i, j+\ell, S)$, then by Fact 27 $Col(i, j+\ell, S) \neq First(U)$, hence after using U at position (i, j) we will not be able to cover position $(i, j+\ell)$, and we know that $j + \ell \leq m$ and the position is not covered yet. ◀

3.3 Computing all non-unary 2D-tile covers

Let $D(n)$ denote the number of natural divisors of a natural number n.

▶ **Fact 30** ([1]). *$D(n) = o(n^{\varepsilon})$ for every constant $\varepsilon > 0$.*

▶ **Corollary 31.** *There are only $\mathcal{O}(n^{\varepsilon})$ possible shapes of 2D-tile covers of S, for $m \leq n$ and any $\varepsilon > 0$.*

▶ **Theorem 32.** *We can compute all 2D-tile covers in time $\mathcal{O}(N^{1+\epsilon})$.*

Proof. For a given candidate U we can in $\mathcal{O}(|U|)$ time check if property $\gamma(U)$ holds, and then use Algorithm 2 or Algorithm 4, each working in $\mathcal{O}(N)$ time. Due to Corollary 31 there are only $\mathcal{O}(N^{\varepsilon})$ candidates for a tile cover, which we can check in $\mathcal{O}(N^{1+\varepsilon})$ total time. ◀

▶ **Observation 33.** *Just like in case of 1D-tile covers (see Theorem 16) covering of a non-unary 2D-string S with its 2D-tile cover U is unique.*

4 Final remarks

We showed that 2D-tile cover problem can be solved in $\mathcal{O}(N^{1+\epsilon})$ time. Our $\mathcal{O}(N)$ time algorithm for 1D-tiles was based on Lemma 12, which says that all 1D-tiles are powers of the smallest primitive one. However it does not work for 2D-tiles, see Figure 9.

We say that $d \times \ell$ 2D-tile cover is primitive iff it is not a horizontal or vertical power of a smaller 2D-tile.

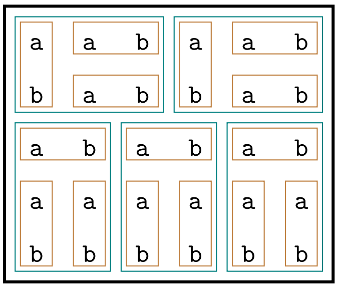

Figure 9 S has three primitive 2D-tile covers of shapes 2×1, 2×3 and 5×6 (S itself). None of them is a horizontal or vertical power of another one (but a smaller one is a 2D-cover of a larger one).

We pose the following conjectures.

▶ **Conjecture 34.** *If a 2D-string has two distinct primitive 2D-tile covers, then one of them is a 2D-tile cover of the other one.*

▶ **Conjecture 35.** *There is a linear time algorithm for computing all 2D-tile covers.*

References

1 Tom M. Apostol. *Introduction to analytic number theory.* Undergraduate Texts in Mathematics, New York-Heidelberg: Springer-Verlag, 1976.

2 Alberto Apostolico, Martin Farach, and Costas S. Iliopoulos. Optimal superprimitivity testing for strings. *Information Processing Letters*, 39(1):17–20, 1991. doi:10.1016/0020-0190(91)90056-N.

3 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.

4 Dany Breslauer. An on-line string superprimitivity test. *Information Processing Letters*, 44(6):345–347, 1992. doi:10.1016/0020-0190(92)90111-8.

5 Panagiotis Charalampopoulos, Jakub Radoszewski, Wojciech Rytter, Tomasz Waleń, and Wiktor Zuba. Computing covers of 2D-strings. In Pawel Gawrychowski and Tatiana Starikovskaya, editors, *32nd Annual Symposium on Combinatorial Pattern Matching, CPM 2021*, volume 191 of *LIPIcs*, pages 12:1–12:20. Schloss Dagstuhl – Leibniz-Zentrum für Informatik, 2021. doi:10.4230/LIPIcs.CPM.2021.12.

6 Maxime Crochemore, Costas S. Iliopoulos, and Maureen Korda. Two-dimensional prefix string matching and covering on square matrices. *Algorithmica*, 20(4):353–373, 1998. doi:10.1007/PL00009200.

7 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.

8 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.

9 Pawel Gawrychowski, Samah Ghazawi, and Gad M. Landau. Lower bounds for the number of repetitions in 2D strings. In Thierry Lecroq and Hélène Touzet, editors, *String Processing and Information Retrieval – 28th International Symposium, SPIRE 2021*, volume 12944 of *Lecture Notes in Computer Science*, pages 179–192. Springer, 2021. doi:10.1007/978-3-030-86692-1_15.

10 Donald E. Knuth, James H. Morris Jr., and Vaughan R. Pratt. Fast pattern matching in strings. *SIAM Journal on Computing*, 6(2):323–350, 1977. doi:10.1137/0206024.

11 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.

12 M. Lothaire. *Combinatorics on words, Second Edition.* Cambridge Mathematical Library. Cambridge University Press, 1997.

13 Alexandru Popa and Andrei Tanasescu. An output-sensitive algorithm for the minimization of 2-dimensional string covers. In *Theory and Applications of Models of Computation – 15th Annual Conference, TAMC 2019*, volume 11436 of *Lecture Notes in Computer Science*, pages 536–549. Springer, 2019. doi:10.1007/978-3-030-14812-6_33.

14 Adrien Thierry. A proof that a word of length n has less than 1.5n distinct squares, 2020. doi:10.48550/ARXIV.2001.02996.