String 2-Covers with No Length Restrictions

Itai Boneh
Reichman University, Herzliya, Israel
University of Haifa, Israel

Shay Golan
Reichman University, Herzliya, Israel
University of Haifa, Israel

Arseny Shur
Bar Ilan University, Ramat Gan, Israel

Abstract

A λ-cover of a string S is a set of strings $\{C_i\}_1^\lambda$ such that every index in S is contained in an occurrence of at least one string C_i. The existence of a 1-cover defines a well-known class of quasi-periodic strings. Quasi-periodicity can be decided in linear time, and all 1-covers of a string can be reported in linear time as well. Since in general it is NP-complete to decide whether a string has a λ-cover, the natural next step is the development of efficient algorithms for 2-covers. Radoszewski and Straszyński [ESA 2020] analysed the particular case where the strings in a 2-cover must be of the same length. They provided an algorithm that reports all such 2-covers of S in time near-linear in $|S|$ and in the size of the output.

In this work, we consider 2-covers in full generality. Since every length-n string has $\Omega(n^2)$ trivial 2-covers (every prefix and suffix of total length at least n constitute such a 2-cover), we state the reporting problem as follows: given a string S and a number m, report all 2-covers $\{C_1, C_2\}$ of S with length $|C_1| + |C_2|$ upper bounded by m. We present an $\tilde{O}(n + \texttt{output})$ time algorithm solving this problem, with `output` being the size of the output. This algorithm admits a simpler modification that finds a 2-cover of minimum length. We also provide an $\tilde{O}(n)$ time construction of a 2-cover oracle which, given two substrings C_1, C_2 of S, reports in poly-logarithmic time whether $\{C_1, C_2\}$ is a 2-cover of S.

2012 ACM Subject Classification Theory of computation → Design and analysis of algorithms

Keywords and phrases Quasi-periodicity, String cover, Range query, Range stabbing

Digital Object Identifier 10.4230/LIPIcs.ESA.2024.31

Related Version *Full Version*: `https://arxiv.org/abs/2405.11475` [8]

Funding *Itai Boneh*: supported by Israel Science Foundation grant 810/21.
Shay Golan: supported by Israel Science Foundation grant 810/21.
Arseny Shur: supported by the ERC grant MPM under the EU's Horizon 2020 Research and Innovation Programme (grant no. 683064) and by the State of Israel through the Center for Absorption in Science of the Ministry of Aliyah and Immigration.

Acknowledgements We are grateful to anonymous reviewers for careful reading and useful comments. Our special thanks for Mikhail Rubinchik and one of the reviewers for independently suggesting a faster algorithm for Lemma 34.

1 Introduction

For a string S, the substring C of S is a *cover* of S if every index of S is covered by an occurrence of C. Since the introduction of covers by Apostolico and Ehrenfeucht [4], many algorithms have been developed for finding covers or variations of covers of a given string. [4] presented an $O(n \log^2 n)$ time algorithm for finding all covers of an input string of length n. It was shown by Moore and Smyth [23, 24] that all covers of a string can be reported in

$O(n)$ time. Li and Smyth [22] further extended this result by showing that the covers of all prefixes of S can be computed in $O(n)$ time. Further works on covers and variants of cover include [1, 9, 18, 7, 2, 3, 15, 6, 25].

A natural generalization of a cover is a λ-cover. A set of strings $\{C_1, C_2, \ldots, C_\lambda\}$ is a λ-cover of S if every index in S is covered by an occurrence of C_i for some $i \in [\lambda]$. The notion of λ-cover was first studied by Cole et al. [11], who proved that it is NP-complete to decide, given a string S and integers k and λ, whether S can be covered by λ substrings of length k. Apparently, relaxing the length restrictions would not ease the problem, but we are unaware of any other published hardness result. Moreover, Guo et al. [16] made an erroneous claim that for a fixed λ and a constant-size alphabet all λ-covers of a string S can be computed in $O(n^2)$ time; the flaw in their analysis was pointed out by Czajka and Radoszewski [12].

In this work, we focus on 2-covers. Radoszewski and Straszyński [26] considered the "balanced" case mentioned above, where the two strings composing the 2-cover have equal length. They proposed an $\tilde{O}(n + \mathsf{output})$-time algorithm reporting all balanced 2-covers of a given length-n string S. They also provide two versions of this algorithm, one of which finds a balanced 2-cover of each possible length and the other determines the shortest balanced 2-cover of S; both versions work in $\tilde{O}(n)$ time. Designing efficient algorithms for the same problems on 2-covers in the general case is posed in [26] as an open problem.

For a 2-cover $\{C_1, C_2\}$, its *length* is $|C_1|+|C_2|$. Following the open problem of Radoszewski and Straszyński, we specify the following problems for 2-covers in the general case:

- All_2-covers(S, m): for a string S, report all 2-covers of length at most m;
- Shortest_2-cover(S): for a string S, find a 2-cover of minimum length;
- 2-cover_Oracle(S): for a string S, build a data structure that answers the queries of the form "do given two substrings of S constitute a 2-cover of S?"

Note that every length-n string S trivially has $\Omega(n^2)$ 2-covers. E.g., if C_1 is a prefix of S, C_2 is a suffix of S, and $|C_1| + |C_2| \geq n$, then $\{C_1, C_2\}$ is a 2-cover of S. The length restriction made in the above formulation of the All_2-covers problem allows one to consider instances with smaller outputs, thus returning the running times of type $\tilde{O}(n + \mathsf{output})$ into the game.

Our contribution. In this work, we solve the three above problems in near linear time.

▶ **Theorem 1.** *There exists an algorithm that solves* All_2-covers(S, m) *in* $O(n \log^5 n + \mathsf{output} \cdot \log^3 n)$ *time for any value of the bound* m.

▶ **Theorem 2.** *There exists an algorithm that solves* Shortest_2-cover(S) *in* $O(n \log^3 n)$ *time.*

▶ **Theorem 3.** *There exists an algorithm that solves* 2-cover_Oracle(S) *in* $O(n \log^5 n)$ *preprocessing time and* $O(\log^3 n)$ *query time.*

Techniques and Ideas. The main idea of our algorithms is the formulation of the property "an index i in S is covered by an occurrence of a substring U of S" in terms of point location. At a high level, each index i is assigned a compactly representable area $\mathcal{A}_i$ in the plane, and every substring U that is *not highly periodic* corresponds to a point p_U in the plane such that $p_U \in \mathcal{A}_i$ if and only if the index i is covered by some occurrence of U. The same idea, implemented in three dimensions instead of two, covers the case of highly periodic substrings.

Given such a geometric representation, our algorithms make use of multi-dimensional range-reporting and range-stabbing data structures to retrieve and organize the areas associated with each index in S. This organization facilitates the computation of a *core* set of the

2-covers, which consists of pairs of strings that are not highly periodic. This set provides a solution to the Shortest_2-cover problem. Besides that, we create the oracle solving the 2-cover_Oracle problem and utilize the core set to finalize our solution to the All_2-covers problem with a small number of queries to this oracle.

Organization. In Section 2 we present notation, auxiliary lemmas, and pre-existing data structures that are used in our algorithms. In Section 3 we formalize and prove the connection between covering an index in a string by an occurrence of a substring and multidimensional point location. In Section 4 we build upon the insights presented in Section 3 to design the 2-cover oracle and prove Theorem 3. Finally, in Section 5 we present the reporting algorithms proving Theorem 2 and Theorem 1 (the latter one executes the oracle). All details omitted due to space constraints can be found in the full version [8].

2 Preliminaries

Here we present definitions, notation, and auxiliary lemmas. The proofs are omitted and can be found in the full version [8].

We assume in this paper that $0 \in \mathbb{N}$. We denote $[x..y] = \{i \in \mathbb{N} \mid x \le i \le y\}$ for any *real* numbers x, y, possibly negative. We also denote $[x] = [1..x]$. The notation output stands for the size of output of a reporting algorithm.

All strings in the paper are over an alphabet $\Sigma = \{1, 2, \dots, O(n^c)\}$ for some constant c. The letters of a string S are indexed from 1 to $|S|$. If $X = S[i..j]$, X is called a *substring* of S (a *prefix* of S if $i = 1$, a *suffix* of S if $j = |S|$, and an empty string if $i > j$). We also say that $S[i..j]$ specifies an *occurrence of X at position i*. If X is a substring of S, then S is a *superstring* of X. A string X that occurs both as a prefix and as a suffix of S is a *border* of S. A string S has *period* ρ if $S[i] = S[i+\rho]$ for all $i \in [|S| - \rho]$. Clearly, S has period ρ if and only if $S[1..|S| - \rho]$ is a border of S. The minimal period of S is denoted by $\mathsf{per}(S)$. Let $\mathsf{per}(S) = \rho$. We say that S is *aperiodic* if $|S| < 2\rho$, *(ρ-)periodic* if $|S| \ge 2\rho$, and *highly (ρ-)periodic* if $|S| \ge 3\rho$. We say that S is short (ρ)-periodic if $|S| \in [2\rho..3\rho - 1]$. Note that if a string X occurs in S at positions i and j, then $|j - i| \ge \mathsf{per}(X)$.

The following two lemmas specify some useful structure of periodic prefixes and borders.

▶ **Lemma 4.** *Periodic prefixes of a length-n string have, in total, $O(\log n)$ different periods.*

▶ **Lemma 5.** *Every string of length n has $O(\log n)$ aperiodic borders and $O(\log n)$ short periodic borders.*

We use well known notion of the longest common prefix (LCP).

▶ **Definition 6.** *For two strings S and T, $\mathsf{LCP}(S,T) = \max\{\ell \mid S[1..\ell] = T[1..\ell]\}$ is the length of their longest common prefix and $\mathsf{LCP}^R(S,T) = \max\{\ell \mid S[|S| - \ell + 1..|S|] = T[|T| - \ell + 1..|T|]\}$ is the length of their longest common suffix.*

Covers. Given a string S, we say that a substring X *covers* an index i if for some indices $j_1, j_2 \in [|S|]$ we have $X = S[j_1..j_2]$ and $i \in [j_1..j_2]$. We also say that the occurrence of X at j_1 covers i. If X covers every $i \in [|S|]$, we call X a 1-*cover* of S. A pair of substrings (X, Y) is said to cover i if X or Y covers i. If a pair (X, Y) covers every $i \in [|S|]$, we call (X, Y) a 2-*cover* of S. (It will be convenient to consider 2-covers as ordered pairs, though the notion of 2-cover is symmetric with respect to X and Y.) We say that (X, Y) is highly periodic if either X or Y is highly periodic, otherwise (X, Y) is non-highly periodic. The following lemma considers a periodic string that covers index i.

▶ **Lemma 7.** *Let S be a string, and let X be a ρ-periodic substring. If the string X covers an index i, then the string $X[1..|X|-\rho]$ ($= X[\rho+1..|X|]$) also covers i.*

Runs. A ρ-periodic substring of S is a *run* if it is not contained in a longer ρ-periodic substring. We use the following lemmas regarding runs.

▶ **Lemma 8.** *Let S be a string, $\rho \in [|S|]$. Every index i in S is covered by at most two ρ-periodic runs and by $O(\log n)$ highly periodic runs.*

▶ **Lemma 9.** *Let S be a string. For every ρ-periodic substring $S[i..j]$, there is a unique ρ-periodic run containing $S[i..j]$.*

▶ **Lemma 10.** *If there is an integer ρ such that $S[x..y]$ is ρ-periodic and $S[x..y+1]$ is not ρ-periodic, then $S[x..y+1]$ is aperiodic.*

▶ **Lemma 11** ([5, Theorem 9]). *The number of runs in any string S is smaller than $|S|$.*

2.1 Range Data Structures

Our algorithms use data structures for *orthogonal range queries*. Such a data structure is associated with a positive integer dimension d and deals with d-dimensional points and d-dimensional ranges. A d-dimensional point is a d-tuple $p = (x_1, x_2, \ldots, x_d)$ and a d-dimensional range is the cartesian product $R = [a_1..b_1] \times [a_2..b_2] \times \ldots \times [a_d..b_d]$ of d ranges. We call a 2-dimensional range a *rectangle* and a 3-dimensional range a *cuboid*. We say that a point p is contained in the range R (denoted by $p \in R$) if $x_i \in [a_i..b_i]$ for every $i \in [d]$.

We make use of the following range data structures.

▶ **Lemma 12** (Range Query Data Structure [28, 10]). *For any integer d, a set P of n points in $\mathbb{R}^d$ can be preprocessed in $O(n \log^{d-1} n)$ time to support the following queries.*
- ***Reporting:*** *Given a d-dimensional range R, output all points in the set $P \cap R$.*
- ***Emptiness:*** *Given a d-dimensional range R, report if $P \cap R = \emptyset$ or not.*

The query time is $O(\log^{d-1} n)$ for Emptiness and $O(\log^{d-1} n + \mathsf{output})$ for Reporting.

▶ **Lemma 13** (Range stabbing queries [10, Theorems 5 and 7]). *For any integer d, a set of d-dimensional ranges $R_1, R_2, \ldots, R_n$ can be preprocessed in $O(n \log^{d-1} n)$ time to support the following queries.*
- ***Stabbing:*** *Given a d-dimensional point p, report all ranges R_i such that $p \in R_i$.*
- ***Existence:*** *Given a d-dimensional point p, report if* **no** *ranges R_i satisfy $p \in R_i$.*

The query time is $O(\log^{d-1} n)$ for Existence and $O(\log^{d-1} n + \mathsf{output})$ for Stabbing.

2.2 Stringology Algorithms and Data Structures

Throughout the paper, we make use of the following string algorithms and data structures.

▶ **Lemma 14** (Pattern Matching [17]). *There exists an algorithm that, given a string T of length n and a string P of length $m \leq n$, reports in $O(n)$ time all the occurrences of P in T.*

▶ **Lemma 15** (LCP_S Data Structure [20, 14]). *There exists a data structure LCP_S that preprocesses an arbitrary string $S \in \Sigma^*$ of length n in $O(n)$ time and supports constant-time queries $\mathsf{LCP}_S(i,j) = \mathsf{LCP}(S[i..n], S[j..n])$ and $\mathsf{LCP}^R_S(i,j) = \mathsf{LCP}^R(S[1..i], S[1..j])$.*

When S is clear from context, we simply write $\mathsf{LCP}(i,j)$ and $\mathsf{LCP}^R(i,j)$.

▶ **Lemma 16** (Internal Pattern Matching (IPM) [19]). *There exists a data structure* IPM_S *that preprocesses an arbitrary string* $S \in \Sigma^*$ *of length* n *in* $O(n)$ *time and supports the following constant-time queries.*

- ***Periodic:*** *given a substring* X*, return* $\mathsf{per}(X)$ *if* X *is periodic, and "aperiodic" otherwise.*
- ***Internal Matching:*** *Given two substrings* X *and* Y *such that* $|Y| = O(|X|)$*, return all occurrences of* X *in* Y *represented as* $O(1)$ *arithmetic progressions.*

▶ **Lemma 17** (Finding all Substrings, see [21]). *There is an algorithm that reports all distinct substrings of a string* $S[1..n]$ *in time* $O(n + \mathsf{output})$.

▶ **Lemma 18** (Finding all Runs [13, Theorem 1.4]). *There is an algorithm that computes all runs of a string* $S \in \Sigma^*$ *of length* n *in* $O(n)$ *time.*

3 Range Characterization of Covering an Index

In this section we translate the property "an index is covered by an occurrence of a given substring" to the language of d-dimensional points and ranges. Then this property can be checked with the queries described in Lemmas 12 and 13. We distinguish between the 2-dimensional case of not highly periodic substrings (Lemma 19) and 3-dimensional case of highly periodic substrings (Lemma 26). Given a point p and a set $\mathcal{R}$ of ranges (both in d dimensions), we slightly abuse the notation, writing $p \in \mathcal{R}$ instead of $p \in \bigcup_{R\in\mathcal{R}} R$.

To present the algorithms that prove these lemmas, we first describe an $O(n\log^2 n)$ time preprocessing phase. Throughout the rest of the paper, we assume that this preprocessing has already been executed.

Preprocessing. The algorithm computes LCP_S data structure of Lemma 15 and IPM_S data structure of Lemma 16. In addition, the algorithm computes all runs of S using Lemma 18. For every $\rho \in [n]$, the algorithm stores all ρ-periodic runs of S in a 3-dimensional range reporting data structure D^ρ_{run} of Lemma 12 as follows. For every ρ-periodic run $S[\ell..r]$, the data structure D^ρ_{run} contains the point $p = (\ell, r, r - \ell + 1)$. By Lemma 11, the total number of runs stored in the structures D^ρ_{run} over all $\rho \in [n]$ is at most n. It follows from Lemmas 12, 15, 16, and 18 that the preprocessing time is $O(n\log^2 n)$.

3.1 The Not Highly Periodic Case

In this section we prove the following lemma.

▶ **Lemma 19.** *Let* $f, i \in [n]$ *be two indices and let* $k \in \mathbb{N}$*. There exists a set* $\mathcal{R}$ *of* $O(1)$ *rectangles such that for any* $\ell, r \in \mathbb{N}$ *with* $\ell+r+1 \in [1.5^k..1.5^{k+1}]$ *the string* $\mathsf{sub} = S[f-\ell..f+r]$ *satisfies the following conditions:*
1. *If* $(\ell, r) \in \mathcal{R}$*, then* sub *covers* i *and* $\mathsf{per}(\mathsf{sub}) \ge \frac{1.5^k}{4}$.
2. *If* sub *covers* i *and is not highly periodic, then* $(\ell, r) \in \mathcal{R}$.

Moreover, $\mathcal{R}$ *can be computed in* $O(\log^2 n)$ *time.*

For $f \in [n]$ and $k \in \mathbb{N}$, let $\mathsf{sub}_{\mathsf{left}} = S[f - \left\lfloor \frac{1.5^k}{2} \right\rfloor ..f]$, and $\mathsf{sub}_{\mathsf{right}} = S[f..f + \left\lfloor \frac{1.5^k}{2} \right\rfloor]$. If an endpoint of a substring is outside S, the substring is undefined.

▶ **Observation 20.** *Let* $f \in [n]$ *be an index and* $k \in \mathbb{N}$*. For every* $\mathsf{sub} = S[f - \ell..f + r]$ *such that* $\ell, r \in \mathbb{N}$ *and* $|\mathsf{sub}| \in [1.5^k..1.5^{k+1}]$*,* sub *is a superstring of either* $\mathsf{sub}_{\mathsf{left}}$ *or* $\mathsf{sub}_{\mathsf{right}}$.

Observation 20 allows us to prove Lemma 19 as follows. First we find a set $\mathcal{R}_1$ that satisfies the conditions of the lemma for all pairs (ℓ, r) such that $\mathsf{sub} = S[f-\ell..f+r]$ is a superstring of $\mathsf{sub}_{\mathsf{right}}$. Similarly, we find a set $\mathcal{R}_2$ for the case where sub is a superstring of $\mathsf{sub}_{\mathsf{left}}$. Then $\mathcal{R}_1 \cup \mathcal{R}_2$ is the set required by the lemma.

In the rest of the section we show how to find the set $\mathcal{R}_1$. (The argument for the set $\mathcal{R}_2$ is similar, so we omit it.) Let us fix f, ℓ, and $r \geq \left\lfloor \frac{1.5^k}{2} \right\rfloor = |\mathsf{sub}_{\mathsf{right}}| - 1$. Let i_{right} denote the starting index of an occurrence of $\mathsf{sub}_{\mathsf{right}}$. We make the following claim.

▷ Claim 21. There exists a rectangle R such that for any $\ell, r \in \mathbb{N}$ with $\ell+r+1 \in [1.5^k..1.5^{k+1}]$ and $r \geq \left\lfloor \frac{1.5^k}{2} \right\rfloor$ the substring $\mathsf{sub} = S[f-\ell..f+r]$ covers the index i **with the occurrence at position** $i_{\mathsf{right}} - \ell$ if and only if $(\ell, r) \in R$. Moreover, R can be computed in $O(1)$ time.

Proof. Let $e_r = \mathsf{LCP}(f, i_{\mathsf{right}})$, $e_\ell = \mathsf{LCP}^R(f, i_{\mathsf{right}})$. Using LCP_S, we compute in $O(1)$ time the rectangle $R = [i_{\mathsf{right}} - i \,..\, e_\ell - 1] \times [i - i_{\mathsf{right}} \,..\, e_r - 1]$ and check the required conditions.

First assume $(\ell, r) \in R$. Since $\ell \leq e_\ell - 1$ and $r \leq e_r - 1$, one has $\mathsf{sub} = S[f-\ell..f+r] = S[i_{\mathsf{right}} - \ell..i_{\mathsf{right}} + r]$, so sub occurs at $i_{\mathsf{right}} - \ell$. Since $\ell \geq i_{\mathsf{right}} - i$ and $r \geq i - i_{\mathsf{right}}$, we have $i \in [i_{\mathsf{right}} - \ell..i_{\mathsf{right}} + r]$, so this occurrence covers i.

Now assume that sub covers i with the occurrence at $i_{\mathsf{right}} - \ell$. Since sub occurs at $i_{\mathsf{right}} - \ell$, one has $\ell \leq e_\ell - 1$ and $r \leq e_r - 1$. Since this occurrence covers i, one also has $i \in [i_{\mathsf{right}} - \ell..i_{\mathsf{right}} - \ell + |\mathsf{sub}| - 1] = [i_{\mathsf{right}} - \ell..i_{\mathsf{right}} + r]$. Then $\ell \geq i_{\mathsf{right}} - i$ and $r \geq i - i_{\mathsf{right}}$, which finally proves $(\ell, r) \in R$. ◁

If sub covers index i, its substring $\mathsf{sub}_{\mathsf{right}}$ must occur close to i. Since $|\mathsf{sub}| \leq 1.5^{k+1}$, if an occurrence of sub at $i_{\mathsf{right}} - \ell$ covers i, then the position i_{right}, at which $\mathsf{sub}_{\mathsf{right}}$ occurs, is inside the range $[i - 1.5^{k+1}..i + 1.5^{k+1}]$. Let $\mathsf{occ}_{\mathsf{right}}$ be the set of all such indices i_{right} from this range. We distinguish between two cases, regarding the period of $\mathsf{sub}_{\mathsf{right}}$.

Case 1: $\mathsf{per}(\mathsf{sub}_{\mathsf{right}}) \geq \frac{1.5^k}{4}$. The following claim is easy.

▷ Claim 22. If $\mathsf{per}(\mathsf{sub}_{\mathsf{right}}) \geq \frac{1.5^k}{4}$, then $|\mathsf{occ}_{\mathsf{right}}| = O(1)$.

Proof. The distance between two consecutive occurrences of $\mathsf{sub}_{\mathsf{right}}$ is at least $\frac{1.5^k}{4}$. Since a range of length $2 \cdot 1.5^{k+1}$ contains $O(1)$ disjoint ranges of length $\frac{1.5^k}{4}$, the claim follows. ◁

Now we build the set $\mathcal{R}_1$. We compute $\mathsf{occ}_{\mathsf{right}}$ in $O(1)$ time using the IPM_S data structure. For every $j \in \mathsf{occ}_{\mathsf{right}}$, we take the rectangle R_j from Claim 21. Let $\mathcal{R}_1 = \{R_j \mid j \in \mathsf{occ}_{\mathsf{right}}\}$. By Claim 22, $|\mathcal{R}_1| = O(1)$. Consider a pair (ℓ, r) such that $\ell + r + 1 \in [1.5^k..1.5^{k+1}]$ and $r \geq \left\lfloor \frac{1.5^k}{2} \right\rfloor$, and let $\mathsf{sub} = S[f-\ell..f+r]$. If $(\ell, r) \in \mathcal{R}_1$, then $(\ell, r) \in R_j$ for some $j \in \mathsf{occ}_{\mathsf{right}}$. Hence by Claim 21 sub covers i with an occurrence at $j - \ell$ and $\mathsf{per}(\mathsf{sub}) \geq \mathsf{per}(\mathsf{sub}_{\mathsf{right}}) \geq \frac{1.5^k}{4}$, as required. Conversely, if sub covers i with an occurrence at j, then $j' = j + \ell$ belongs to $\mathsf{occ}_{\mathsf{right}}$. Then by Claim 21 $(\ell, r) \in R_{j'}$ and thus $(\ell, r) \in \mathcal{R}_1$. This concludes the proof of Lemma 19 in the case $\mathsf{per}(\mathsf{sub}_{\mathsf{right}}) \geq \frac{1.5^k}{4}$.

Case 2: $\rho = \mathsf{per}(\mathsf{sub}_{\mathsf{right}}) < \frac{1.5^k}{4}$. Let $\mathsf{run}_f = S[f - \ell^f_{\mathsf{run}}..f + r^f_{\mathsf{run}}]$ be the ρ-periodic run containing $S[f..f + |\mathsf{sub}_{\mathsf{right}}| - 1]$ (such a run exists by Lemma 9). Consider a pair (ℓ, r) such that $\ell + r + 1 \in [1.5^k..1.5^{k+1}]$ and $r \geq \left\lfloor \frac{1.5^k}{2} \right\rfloor$, and let the string $\mathsf{sub} = S[f-\ell..f+r]$ be not highly periodic. Then $\mathsf{per}(\mathsf{sub}) > \frac{1.5^k}{3} > \rho$. Hence sub is not a substring of run_f, which means that either $\ell > \ell^f_{\mathsf{run}}$ or $r > r^f_{\mathsf{run}}$. Below we assume $r > r^f_{\mathsf{run}}$; the other case is symmetric. We first observe that this inequality guarantees that $\mathsf{per}(\mathsf{sub})$ is big enough.

▷ Claim 23. For every $\mathsf{sub} = S[f-\ell..f+r]$ with $r > r^f_{\mathsf{run}}$ one has $\mathsf{per}(\mathsf{sub}) \geq \frac{1.5^k}{4}$.

Proof. The substring $S[f..f+r^f_{\mathsf{run}}]$ is ρ-periodic and $u = S[f..f+r^f_{\mathsf{run}}+1]$ is not (otherwise, run_f is not a run). By Lemma 10, u is aperiodic. Then $\mathsf{per}(u) > \frac{|u|}{2} \geq \frac{1.5^k}{4}$. It remains to note that u is a substring of sub. ◁

Let $\mathsf{sub}^{\rightarrow}_{\mathsf{right}} = S[f..f+r^f_{\mathsf{run}}]$. Note that $\mathsf{sub}^{\rightarrow}_{\mathsf{right}}$ is a ρ-periodic suffix of the ρ-periodic run run^f and sub contains $\mathsf{sub}^{\rightarrow}_{\mathsf{right}}$ followed by a letter that breaks the period ρ. This means that if sub covers i, then S contains, close to i, a ρ-periodic run with the suffix $\mathsf{sub}^{\rightarrow}_{\mathsf{right}}$. Let us say that a ρ-periodic run $S[a_{\mathsf{run}}\,..\,b_{\mathsf{run}}]$ is *close to* i if $a_{\mathsf{run}} \leq i + 1.5^{k+1}$ and $b_{\mathsf{run}} \geq i - 1.5^{k+1}$. Clearly, if sub covers i, it contains the suffix $\mathsf{sub}^{\rightarrow}_{\mathsf{right}}$ of a run close to i.

Let $\mathsf{Run}_{\mathsf{close}}$ be the set of ρ-periodic runs close to i with length at least $|\mathsf{sub}^{\rightarrow}_{\mathsf{right}}|$.

▷ Claim 24. $|\mathsf{Run}_{\mathsf{close}}| = O(1)$. Moreover, $\mathsf{Run}_{\mathsf{close}}$ can be computed in $O(\log^2 n)$ time.

Proof. Assume that $\mathsf{Run}_{\mathsf{close}}$ is ordered by the positions of runs. Each of these runs has length at least $\frac{1.5^k}{2}$ and any two ρ-periodic runs overlap by less than $\rho < \frac{1.5^k}{4}$ positions. Then the positions of any two consecutive runs from $\mathsf{Run}_{\mathsf{close}}$ differ by more than $\frac{1.5^k}{4}$ and any two non-consecutive runs are disjoint. Since the first run ends no later than the position $i - 1.5^{k+1}$ by definition of being close to i, the third and all subsequent runs start after this position. Again by definition, all runs start before the position $i + 1.5^{k+1}$. The range $[i-1.5^{k+1}..i+1.5^{k+1}]$ contains $O(1)$ positions such that any two of them differ by more than $\frac{1.5^k}{4}$. Hence we get $|\mathsf{Run}_{\mathsf{close}}| = O(1)$.

Querying D^{ρ}_{run} with the range $[-\infty..i+1.5^{k+1}] \times [i-1.5^{k+1}..\infty] \times [|\mathsf{sub}^{\rightarrow}_{\mathsf{right}}|..\infty]$ we get all ρ-periodic runs that are close to i (due to the first two coordinates) and have length at least $|\mathsf{sub}^{\rightarrow}_{\mathsf{right}}|$ (due to the last coordinate); i.e., what we get is $\mathsf{Run}_{\mathsf{close}}$. The query time is $O(\log^2 n)$ by Lemma 12. ◁

Now we construct the set $\mathcal{R}_1$. We query D^{ρ}_{run} with $[-\infty..f] \times [f + \left\lfloor \frac{1.5^k}{2} \right\rfloor..\infty] \times [-\infty..\infty]$ to get the unique ρ-periodic run $\mathsf{run}_f = [f - \ell^f_{\mathsf{run}}..f + r^f_{\mathsf{run}}]$ containing the substring $\mathsf{sub}_{\mathsf{right}} = S[f..f + \left\lfloor \frac{1.5^k}{2} \right\rfloor]$. Then we compute the $O(1)$-size set $\mathsf{Run}_{\mathsf{close}}$ (Claim 24). For every $\mathsf{run} \in \mathsf{Run}_{\mathsf{close}}$ we check, with an LCP query, whether $\mathsf{sub}^{\rightarrow}_{\mathsf{right}}$ is a suffix of run. If yes, we compute the position i_{right} of this suffix from the parameters of the run. Since i_{right} is the position of an occurrence of $\mathsf{sub}_{\mathsf{right}}$, we apply Claim 21 to obtain a rectangle $R = [\ell_1, \ell_2] \times [r_1, r_2]$. Since in our argument we assume $r > r^f_{\mathsf{run}}$, we replace r_1 with $\max\{r_1, r^f_{\mathsf{run}} + 1\}$. If the range for r remains nonempty, we denote the obtained rectangle by R_{run}.

Let $\mathcal{R}_{\mathsf{right}} = \{R_{\mathsf{run}} \mid \mathsf{run} \in \mathsf{Run}_{\mathsf{close}}\}$. In a symmetric way, we consider the case $\ell > \ell^f_{\mathsf{run}}$ and build the set $\mathcal{R}_{\mathsf{left}}$. Finally we set $\mathcal{R}_1 = \mathcal{R}_{\mathsf{right}} \cup \mathcal{R}_{\mathsf{left}}$. The time complexity is dominated by $O(1)$ queries to D^{ρ}_{run}, which take $O(\log^2 n)$ by Lemma 12.

Now consider a pair (ℓ, r) such that $\ell + r + 1 \in [1.5^k..1.5^{k+1}]$ and $r \geq \left\lfloor \frac{1.5^k}{2} \right\rfloor$, and let $\mathsf{sub} = S[f-\ell..f+r]$. If $(\ell, r) \in \mathcal{R}_1$, then (ℓ, r) belongs to some rectangle from $\mathcal{R}_{\mathsf{right}}$ or $\mathcal{R}_{\mathsf{left}}$; these cases are symmetric, so let this rectangle be $R_{\mathsf{run}} \in \mathcal{R}_{\mathsf{right}}$, where $\mathsf{run} = [a_{\mathsf{run}}..b_{\mathsf{run}}]$. Hence by Claim 21 sub covers i with an occurrence at $b_{\mathsf{run}} - r^f_{\mathsf{run}} - \ell$. We also have $\mathsf{per}(\mathsf{sub}) \geq \frac{1.5^k}{4}$ by Claim 23. Conversely, if sub is not highly periodic, then either $r > r^f_{\mathsf{run}}$ or $\ell > \ell^f_{\mathsf{run}}$. Without loss of generality, let $r > r^f_{\mathsf{run}}$. Now if sub covers i with an occurrence at j, then there is an occurrence of $\mathsf{sub}^{\rightarrow}_{\mathsf{right}}$ at $j' = j + \ell$ that is a suffix of a ρ-periodic run run. Then by Claim 21 we have $(\ell, r) \in R_{\mathsf{run}}$ and thus $(\ell, r) \in \mathcal{R}_1$. Thus, we finished the proof of Lemma 19.

3.2 The Highly Periodic Case

In this section we prove Lemma 26, which is the analog of Lemma 19 for periodic strings.

We begin with more notation. Let u be a ρ-periodic string having a substring v of length ρ. Since $\rho = \mathsf{per}(u)$, the positions of any two occurrences of v in u differ by a multiple of ρ. Then there exist unique integers $t_1, t_2 \in [0..\rho-1]$ and $q \in \mathbb{N}$ such that $u = v[\rho{-}t_1{+}1..\rho]v^q v[1..t_2]$. We abbreviate this representation as $u = v^{[t_1;q;t_2]}$.

▶ **Observation 25.** *Let $u = v^{[t_1;q;t_2]}$. The numbers t_1, t_2, and q can be computed in $O(1)$ time given $|u|, |v|$, and the position of any occurrence of v in u.*

For a ρ-periodic substring $\mathsf{sub} = S[f-\ell..f+r]$, we define its *root* by

$$\mathsf{root} = \begin{cases} S[f..f+\rho-1], & \text{if } r \geq \rho - 1, \\ S[f-\rho..f-1], & \text{otherwise.} \end{cases}$$

Thus, $\mathsf{sub} = \mathsf{root}^{[t_\ell;q_{\ell,r};t_r]}$ for some unique integers $t_\ell, t_r \in [0..\rho-1]$ and $q_{\ell,r} > 0$.

▶ **Lemma 26.** *Let $f, i \in [n]$ be two indices and let $\rho \in [n]$. There exists a set $\mathcal{C}$ of $O(1)$ cuboids such that every highly ρ-periodic substring of the form* $\mathsf{sub} = S[f-\ell..f+r]$ *with $\ell, r \in \mathbb{N}$ satisfies the following:* sub *covers index i if and only if $(t_\ell, t_r, q_{\ell,r}) \in \mathcal{C}$. Moreover, $\mathcal{C}$ can be computed in $O(\log^2 n)$ time.*

Let $\mathsf{sub} = S[f-\ell..f+r]$ be highly ρ-periodic. By Lemma 9, each occurrence of sub is contained in a unique ρ-periodic run. By Lemma 8, there are at most two such runs containing i (say, run_1 and run_2). Hence if sub covers i, it does so with an occurrence contained either in run_1 or in run_2. Then Lemma 26 follows from Lemma 27 below: we query D^ρ_{run} to get run_1 and run_2 (in $O(\log^2 n)$ time by Lemma 12), take the sets $\mathcal{C}_1$ and $\mathcal{C}_2$ given by Lemma 27 for run_1 and run_2 respectively, and let $\mathcal{C} = \mathcal{C}_1 \cup \mathcal{C}_2$.

▶ **Lemma 27.** *Let* $\mathsf{run} = S[a_{\mathsf{run}}..b_{\mathsf{run}}]$ *be a ρ-periodic run containing i. There is a set $\mathcal{C}$ of $O(1)$ cuboids such that* sub *covers i* ***with an occurrence fully contained in*** run *if and only if $(t_\ell, t_r, q_{\ell,r}) \in \mathcal{C}$. Moreover, the set $\mathcal{C}$ can be computed in $O(1)$ time.*

In the rest of the section we describe the algorithm computing the set $\mathcal{C}$ of Lemma 27.

First the algorithm checks the length of run. Since sub is highly ρ-periodic, $\mathsf{sub} \geq 3\rho$. If $|\mathsf{run}| < 3\rho$, then sub has no occurrences in run. Hence in this case $\mathcal{C} = \varnothing$. Next, the algorithm verifies if root is a substring of run. Due to ρ-periodicity of run, it is sufficient to check for an occurrence of root the prefix of run having length $2\rho = 2|\mathsf{root}|$. By Lemma 16, this check can be done in $O(1)$ time with the IPM_S data structure. If root is not a substring of run, then once again sub has no occurrences in run and so $\mathcal{C} = \varnothing$.

From now on, we assume that $|\mathsf{run}| \geq 3\rho$ and run contains an occurrence of root. Then the algorithm computes, in $O(1)$ time by Observation 25, the parameters of the representation $\mathsf{run} = \mathsf{root}^{[t_\ell^{\mathsf{run}};q_{\mathsf{run}};t_r^{\mathsf{run}}]}$. One has $q_{\mathsf{run}} \geq 2$ since $|\mathsf{run}| \geq 3\rho$. We recall that $\mathsf{sub} = \mathsf{root}^{[t_\ell;q_{\ell,r};t_r]}$.

Note that sub covers index i of S with an occurrence contained in run if and only if it covers the index $j = i - a_{\mathsf{run}} + 1$ *of the string* run. In order to build the set $\mathcal{C}$, we describe, in Claim 30, a set of necessary and sufficient conditions for sub to cover an index j of run. We denote $b_\ell = [t_\ell > t_\ell^{\mathsf{run}}]$, $b_r = [t_r > t_r^{\mathsf{run}}]$ (Iverson bracket notation). We need two auxiliary claims.

▷ Claim 28. Let sub occur in run. If x is the starting index of its leftmost occurrence and y is the ending index of its rightmost occurrence, then $x = t_\ell^{\mathsf{run}} - t_\ell + 1 + b_\ell\rho$, $y = t_\ell^{\mathsf{run}} + (q_{\mathsf{run}} - b_r)\rho + t_r$, and $[x..y]$ is exactly the set of indices covered by sub in run.

Proof. We prove the formula for x as the argument for y is similar. Since run is ρ-periodic, we have $x \in [1..\rho]$ (otherwise, there is another occurrence at position $x-\rho$). Since root occurs in sub at position $t_\ell+1$, run has a matching occurrence of root at position $t_\ell+x \in [t_\ell+1..t_\ell+\rho]$. The occurrences of root in run are at positions $t_\ell^{\mathsf{run}}+1, t_\ell^{\mathsf{run}}+\rho+1, \ldots$, so exactly one of them starts in the range $[t_\ell+1..t_\ell+\rho]$. If $t_\ell \le t_\ell^{\mathsf{run}}$, one has $t_\ell^{\mathsf{run}}+1 \in [t_\ell+1..t_\ell+\rho]$. Therefore, we have $t_\ell^{\mathsf{run}}+1 = t_\ell + x$, implying $x = t_\ell^{\mathsf{run}} - t_\ell + 1 + 0\cdot\rho$ as required. Similarly, if $t_\ell > t_\ell^{\mathsf{run}}$, one has $t_\ell^{\mathsf{run}}+\rho+1 \in [t_\ell+1..t_\ell+\rho]$. Then $t_\ell^{\mathsf{run}}+\rho+1 = t_\ell+x$, which implies $x = t_\ell^{\mathsf{run}} - t_\ell + 1 + 1\cdot\rho$ as required.

Since run is ρ-periodic, the positions of any two consecutive occurrences of sub in run differ by ρ (see Figure 1). As $|\mathsf{sub}| > \rho$, the indices in run covered by occurrences of sub form a single range from the first index of the leftmost occurrence (i.e., x) to the last index of the rightmost occurrence (i.e., y). ◁

▷ Claim 29. The string sub occurs in run if and only if $q_{\ell,r} \le q_{\mathsf{run}} - b_\ell - b_r$.

Proof. Let sub occur in run. By Claim 28, its leftmost occurrence is at $x = d_\ell^{\mathsf{run}} - t_\ell + 1 + b_\ell\rho$ and its rightmost occurrence ends at $y = t_\ell^{\mathsf{run}} + (q_{\mathsf{run}} - b_r)\rho + t_r$. Clearly, we have the inequality $y - x + 1 \ge |\mathsf{sub}| = t_\ell + q_{\ell,r}\rho + t_r$, which is equivalent to $q_{\ell,r} \le q_{\mathsf{run}} - b_\ell - b_r$.

For the converse, we assume that this inequality holds and show that sub occurs in run at position $x = t_\ell^{\mathsf{run}} - t_\ell + 1 + b_\ell\rho$. Since sub and run are both ρ-periodic and share the substring root of length ρ, it suffices to prove that $|\mathsf{run}[x..|\mathsf{run}|]| \ge |\mathsf{sub}|$. Observing that $b_r\rho \ge t_r - t_r^{\mathsf{run}}$, we obtain

$$\begin{aligned}|\mathsf{run}[x..|\mathsf{run}|]| = q_{\mathsf{run}}\rho + t_\ell^{\mathsf{run}} + t_r^{\mathsf{run}} - x + 1 &= (q_{\mathsf{run}} - b_\ell)\rho + t_\ell + t_r^{\mathsf{run}} \\ &\ge (q_{\mathsf{run}} - b_\ell - b_r)\rho + t_\ell + t_r \ge q_{\ell,r}\rho + t_\ell + t_r = |\mathsf{sub}|,\end{aligned}$$

as required. ◁

▷ Claim 30. The string $\mathsf{sub} = \mathsf{root}^{[t_\ell;q_{\ell,r};t_r]}$ covers index j in $\mathsf{run} = \mathsf{root}^{[t_\ell^{\mathsf{run}};q_{\mathsf{run}};t_r^{\mathsf{run}}]}$ if and only if one of the following mutually exclusive conditions holds:

1. $t_\ell \le t_\ell^{\mathsf{run}}$, $t_r \le t_r^{\mathsf{run}}$, $q_{\ell,r} \le q_{\mathsf{run}}$, and $j \in [t_\ell^{\mathsf{run}} - t_\ell + 1 \,..\, t_\ell^{\mathsf{run}} + q_{\mathsf{run}}\rho + t_r]$;
2. $t_\ell > t_\ell^{\mathsf{run}}$, $t_r \le t_r^{\mathsf{run}}$, $q_{\ell,r} \le q_{\mathsf{run}} - 1$, and $j \in [t_\ell^{\mathsf{run}} - t_\ell + 1 + \rho \,..\, t_\ell^{\mathsf{run}} + q_{\mathsf{run}}\rho + t_r]$;
3. $t_\ell \le t_\ell^{\mathsf{run}}$, $t_r > t_r^{\mathsf{run}}$, $q_{\ell,r} \le q_{\mathsf{run}} - 1$, and $j \in [t_\ell^{\mathsf{run}} - t_\ell + 1 \,..\, t_\ell^{\mathsf{run}} + (q_{\mathsf{run}} - 1)\rho + t_r]$;
4. $t_\ell > t_\ell^{\mathsf{run}}$, $t_r > t_r^{\mathsf{run}}$, $q_{\ell,r} \le q_{\mathsf{run}} - 2$, and $j \in [t_\ell^{\mathsf{run}} - t_\ell + 1 + \rho \,..\, t_\ell^{\mathsf{run}} + (q_{\mathsf{run}} - 1)\rho + t_r]$.

Proof. If sub covers index j, then sub occurs in run. Hence Claim 29 implies the inequalities and Claim 28 implies the range for each of conditions 1–4.

For the converse, if one of the conditions 1–4 is true, then sub indeed occurs in run according to Claim 29. Then again Claim 28 implies the range of indices covered by occurrences of sub. As j belongs to this range, it is covered. ◁

The algorithm builds the set $\mathcal{C}$ by running through conditions 1–4 of Claim 30. If $j = i - a_{\mathsf{run}}+1$ belongs to the range from a condition, the algorithm adds to $\mathcal{C}$ the cuboid defined by the inequalities listed in this condition; otherwise, it does nothing. The cuboids for the conditions 1, 2, 3, and 4 are, respectively, $[0..t_\ell^{\mathsf{run}}] \times [0..t_r^{\mathsf{run}}] \times [1..q_{\mathsf{run}}]$; $[t_\ell^{\mathsf{run}}+1..\rho-1] \times [0..t_r^{\mathsf{run}}] \times [1..q_{\mathsf{run}}-1]$; $[0..t_\ell^{\mathsf{run}}] \times [t_r^{\mathsf{run}}+1..\rho-1] \times [1..q_{\mathsf{run}}-1]$; $[t_\ell^{\mathsf{run}}+1..\rho-1] \times [t_r^{\mathsf{run}}+1..\rho-1] \times [1..q_{\mathsf{run}}-2]$.

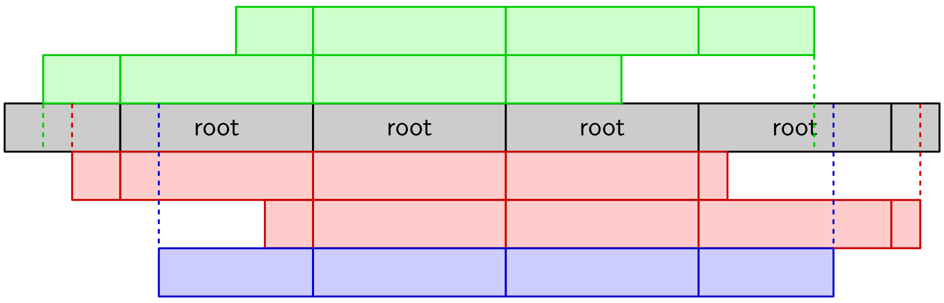

Figure 1 Occurrences of sub in run (Claims 28 and 30). The grey strip is run, occurrences of sub are shown as color strips (one color for one substring sub). The substrings drawn red, green, and blue realize, respectively, conditions 1, 3, and 4 of Claim 30. Dash lines show ranges covered by sub in each case.

Correctness. Let sub cover i with an occurrence contained in run. Then sub covers the index $j = i - a_{\mathsf{run}} + 1$ in run. By Claim 30, the triple $(t_\ell, t_r, q_{\ell,r})$ satisfies one of conditions 1–4, say, condition N. In particular, j belongs to the interval of condition N. Then the algorithm built a cuboid C from the inequalities of condition N such that $(t_\ell, t_r, q_{\ell,r}) \in C$. Conversely, if $(t_\ell, t_r, q_{\ell,r}) \in C$, where C was built from condition N of Claim 30, then j belongs to the interval of condition N. Hence condition N holds; by Claim 30, sub covers the index j in run, and thus covers i in S.

As the time complexity is straightforward, Lemma 27, and then Lemma 26, is proved.

4 2-Covers Oracle

In this section, we present a solution to the 2-cover_Oracle problem (Theorem 3). The preliminary part of the solution is common to all three problems.

Every 2-cover of S contains a prefix and a suffix of S. Respectively, each 2-cover has one of two types (see [26]): a prefix-suffix 2-cover (*ps-cover*) consists of a prefix of S and a suffix of S while in a border-substring 2-cover (*bs-cover*) one string is a border of S. We process these two cases separately.

Let (U_1, U_2) be a pair of substrings. Lemmas 19 and 26 allow us to express each predicate "U_j covers index i" as $p_j \in \mathcal{R}^i_j$, where p_j is a point and $\mathcal{R}^i_j$ is a set of $O(1)$ ranges in d_j dimensions. Then the predicate "(U_1, U_2) is a 2-cover" is expressed by the 2CNF formula $\bigwedge_{i=1}^n (p_1 \in \mathcal{R}^i_1 \vee p_2 \in \mathcal{R}^i_2)$. We answer the instances of this predicate with a new data structure based on rectangle stabbing (Lemma 13).

▶ **Lemma 31** (2-CNF Range Data Structure). *Let d_ℓ, d_r be integer constants and let* Pairs $=$ $\{(\mathcal{L}_1, \mathcal{R}_1), (\mathcal{L}_2, \mathcal{R}_2), \ldots, (\mathcal{L}_n, \mathcal{R}_n)\}$ *be a set of pairs such that for every $i \in [n]$, $\mathcal{L}_i$ is a set of $O(1)$ d_ℓ-dimensional orthogonal ranges and $\mathcal{R}_i$ is a set of $O(1)$ d_r-dimensional orthogonal ranges. The set* Pairs *can be preprocessed in $O(n \log^{d_\ell + d_r - 1} n)$ time to a data structure that supports the following query in $O(\log^{d_\ell + d_r - 1} n)$ time:*

- query(p_ℓ, p_r)*: for a d_ℓ-dimensional point p_ℓ and a d_r-dimensional point p_r, decide if for every $i \in [n]$ either $p_\ell \in \mathcal{L}_i$ or $p_r \in \mathcal{R}_i$.*

In order to prove Lemma 31, we need an auxiliary statement.

▶ **Lemma 32** (Inverse of Ranges). *Let $\mathcal{R}$ be a set of $O(1)$ d-dimensional ranges, where d is an integer constant. There is a set $\overline{\mathcal{R}}$ of $O(1)$ d-dimensional ranges such that $\bigcup_{R\in\overline{\mathcal{R}}} R = [-\infty..\infty]^d \setminus \bigcup_{R\in\mathcal{R}} R$. Moreover, the set $\overline{R}$ can be computed in $O(1)$ time given $\mathcal{R}$.*

Proof. We present a proof for a set of 2-dimensional ranges, i.e., rectangles. This proof can be easily generalized to any constant dimension. For every rectangle $R \in \mathcal{R}$, say, $R = [x_1..x_2] \times [y_1..y_2]$, consider extensions of its sides, which are the lines $x = x_1$, $x = x_2$, $y = y_1$ and $y = y_2$. The extensions of the sides of all rectangles in $\mathcal{R}$ partition the plane into $O(|\mathcal{R}|^2) = O(1)$ rectangles. Every rectangle in this partition is either contained in a rectangle of R, or is disjoint from all rectangles of R. The set of rectangles in the partition disjoint from all rectangles of R satisfies the claim. Moreover, this set can be computed in $O(1)$ time straightforwardly. ◀

Proof of Lemma 31. Let $d = d_\ell + d_r$. We build a set $\mathcal{B}$ of d-dimensional ranges, processing each pair $(\mathcal{L}_i, \mathcal{R}_i)$ as follows. We start with the set B_i of d-dimensional ranges defined by

$$B_i = \{L \times [-\infty..\infty]^{d_r} \mid L \in \mathcal{L}_i\} \cup \{[-\infty..\infty]^{d_\ell} \times R \mid R \in \mathcal{R}_i\}.$$

As $|B_i| = |\mathcal{L}_i| + |\mathcal{R}_i| = O(1)$, we apply Lemma 32 to get, in $O(1)$ time, its inverse set of ranges $\overline{B_i}$, which is also of size $O(1)$. Now let $\mathcal{B} = \bigcup_{i\in[n]} \overline{B_i}$.

We preprocess the set $\mathcal{B}$ into a range stabbing data structure (Lemma 13). To answer $\mathsf{query}(p_\ell, p_r)$, where $p_\ell = (x_1, \ldots, x_{d_\ell})$ and $p_r = (y_1, \ldots, y_{d_r})$, we perform the Existence query to this structure with the d-dimensional point $p = (x_1, \ldots, x_{d_\ell}, y_1, \ldots, y_{d_r})$ and report the *negation* of the obtained answer. The required time complexities follow from Lemma 13.

Correctness. We need to prove that $p \notin \bigcup_{B\in\mathcal{B}} B$ if and only if for every $i \in [n]$ either $p_\ell \in L$ for some $L \in \mathcal{L}_i$ or $p_r \in R$ for some $R \in \mathcal{R}_i$.

Let $p \notin \bigcup_{B\in\mathcal{B}} B$ and let $i \in [n]$. By definition of $\mathcal{B}$, $p \notin \bigcup_{B\in\overline{B_i}} B$. Then $p \in \bigcup_{B\in B_i} B$. By definition of B_i, this implies that $p_\ell \in L$ for some $L \in \mathcal{L}_i$ of $p_r \in R$ for some $R \in \mathcal{R}_i$.

For the converse note that if $p_\ell \in L$ for some $L \in \mathcal{L}_i$ of $p_r \in R$ for some $R \in \mathcal{R}_i$, then $p \in B$ for some $B \in B_i$ and hence $p \notin B'$ for all $B' \in \overline{B_i}$. If this is the case for all i, then $p \notin \bigcup_{B\in\mathcal{B}} B$ by definition. ◀

As Lemma 19 refers to particular ranges and Lemma 26 refers to particular periods, we partition substrings into groups and build a separate 2CNF data structure for each pair of groups. For prefixes, suffixes, and borders, we have $O(\log n)$ periods (Lemma 4) and thus $O(\log n)$ groups of highly periodic prefixes (suffixes, borders). The remaining prefixes (suffixes) form $O(\log n)$ groups associated with length ranges $[1.5^k..1.5^{k+1}]$ for some k. There are $O(\log n)$ remaining borders (Lemma 5), so each of them forms a separate group. For each group of borders we choose a fixed position f. Highly periodic substrings containing f form $O(\log n)$ groups (Lemma 8); the other are grouped according to $O(\log n)$ length ranges. Therefore, in total we build $O(\log^2 n)$ 2CNF data structures for ps-covers and bs-covers.

Effective dimension. A direct implementation of Lemmas 19 and 26 leads to the 2CNF structures of dimension 4 to 6. Let us show how to lower the dimension. For any group pref of prefixes we take $f = 1$. Then in Lemma 19 all points have the form $(0, r)$. So we have *fixed* first coordinate and *variable* second coordinate. In Lemma 26, one has $\mathsf{root} = S[1..\rho]$, and thus all points have the form $(0, d_r, q_r)$ with two variable coordinates. For groups of suffixes we take $f = n$ and symmetrically get the points of the form $(\ell, 0)$ or $(d_\ell, 0, q_\ell)$. Since borders are simultaneously prefixes and suffixes, we get two fixed coordinates in the corresponding points. (Assuming $f = 1$, a group consisting of a single border U, has the point $(0, |U|)$; the group bor of highly ρ-periodic borders has the points $(0, d_r, q)$, where the remainder $d_r = |U| \bmod \rho$ is also fixed: it is the same for all $U \in \mathsf{bor}$.) Finally, for general substrings all coordinates are variable. The *effective dimension* of a point is the number of its variable coordinates. Given a pair of groups of substrings, where the first (second) group has points of *effective* dimension d_1 (respectively, d_2), we construct for them the $\mathsf{2CNF}$ structure of dimension $d_1 + d_2$. In order to do this, we replace each involved range with its projection onto variable coordinates.

Building an oracle. Given a group pref of not highly periodic (resp., highly periodic) prefixes, we apply Lemma 19 (resp., Lemma 26) for every $i \in [n]$. Let $\mathcal{L}_1, \ldots \mathcal{L}_n$ be the projections of the obtained ranges onto variable coordinates. A group suff of suffixes is processed in the same way, resulting in the ranges $\mathcal{R}_1, \ldots, \mathcal{R}_n$. Then we apply Lemma 31, constructing the $\mathsf{2CNF}$ structure over the set $\mathsf{Pairs} = \{(\mathcal{L}_1, \mathcal{R}_1), \ldots, (\mathcal{L}_n, \mathcal{R}_n)\}$. This $\mathsf{2CNF}$ thus represents the set $\mathsf{pref} \times \mathsf{suff}$ of pairs of substrings. We also memorize the values of fixed coordinates. Iterating over all pairs of prefix and suffix groups, we obtain the ps-cover part of the oracle.

Given a group bor of borders, we first determine the reference position f_{bor} for groups of substrings and store it. If $\mathsf{bor} = \{U\}$, we use Lemma 14 to find all occurrences of U in S and choose f_{bor} to be any position not covered by U; if there is no such position, i.e., if U is a 1-cover, we set $f_{\mathsf{bor}} = \infty$. If bor is a group of highly ρ-periodic borders, we run a binary search on it, finding the shortest border U that is not a 1-cover. Then we choose a position f_{bor} not covered by U. Additionally, we store $|U|$. If U does not exist, we set $f_{\mathsf{bor}} = \infty$. After determining f_{bor}, and only if it is finite, we build a $\mathsf{2CNF}$ structure similar to the prefix-suffix case. Iterating over all pairs of border and substring groups (for the latter, we fix $f = f_{\mathsf{bor}}$), we obtain the bs-cover part of the oracle.

The time complexity of the construction is dominated by building $O(\log^2 n)$ $\mathsf{2CNF}$ structures, each of dimension at most 4 (in the case where both groups consist of highly periodic strings). By Lemma 31, we get the required $O(n \log^5 n)$ time bound.

Querying an oracle. Given a pair (U_1, U_2) of substrings of S, the oracle decides with LCP queries, to which of the cases (prefix-suffix, border-substring, both, or neither) this pair can be attributed, and proceeds accordingly. For prefix, suffix, or border, the oracle finds its group deciding high periodicity with a query to IPM_S (Lemma 16). In the prefix-suffix case the oracle then creates points for U_1 and U_2, "trims" them by dropping fixed coordinates, and queries with this pair of trimmed points the $\mathsf{2CNF}$ structure built for the set $\mathsf{pref} \times \mathsf{suff}$, where the groups pref and suff contain U_1 and U_2 respectively.

Consider the border-substring case (let U_1 be the border). After determining the group bor of U_1, we check f_{bor}. If $f_{\mathsf{bor}} = \infty$, the oracle returns True since U_1 is a 1-cover. The same applies for the case where f_{bor} is finite, bor is highly periodic, and U_1 is shorter than the saved length $|U|$. Otherwise, we create the point for U_1, drop its fixed coordinates, and create the point for U_2 using $f = f_{\mathsf{bor}}$. Then we query with the obtained pair of points the $\mathsf{2CNF}$ structure built for the pair $\mathsf{bor} \times \mathsf{sub}_f$, where the group sub_f contains U_2.

Finally, the oracle returns True if it met a condition for "True" in the border-substring case or if some query made to a $\mathsf{2CNF}$ structure returned True. Otherwise, the oracle returns False. The query time is dominated by $O(1)$ queries to $\mathsf{2CNF}$ structures, each of dimension at most 4. By Lemma 31, we get the required $O(\log^3 n)$ time bound.

As a result, we proved Theorem 3. The omitted details can be found in [8].

5 Reporting 2-Covers

A possible, but in general highly inefficient, approach to the All_2-covers and Shortest_2-cover problems is to construct the oracle of Theorem 3 and query it with every pair (of substrings) that can be in the answer. In this section, we describe our approach to achieve near-linear running time. In a high level, we build a fast reporting procedure for "simple" cases and use its answer to determine the rest of the output with a small number of oracle queries. To rule out the trivial situation, we assume that all 2-covers containing a 1-cover are already reported just by listing the 1-covers.

We call a 2-cover (X, Y) *core* if both substrings X and Y are not highly periodic. This means that the 2CNF structure for their groups is built by using only Lemma 19, and thus is 2-dimensional. In particular, a core cover is associated with a 2-dimensional point (x, y). Note that 2-dimensional 2CNF structures represent all core 2-covers (and may represent some non-core 2-covers as certain highly periodic substrings pass the restriction on periods in statement 1 of Lemma 19). The shortest 2-cover is core in view of Lemma 7.

On the ground level, a d-dimensional 2CNF structure stores a set $\mathcal{R}$ of $O(n)$ d-dimensional ranges and checks whether a d-dimensional point, sent as a query, is outside all rectangles. Such a view inspires the following definition for the case $d = 2$ we are interested in.

▶ **Definition 33** (Free Point). *Let $\mathcal{R}$ be a set of rectangles with corners in $[n]^2$. A point $p \in [n]^2$ is $\mathcal{R}$-free if $p \notin R$ for every $R \in \mathcal{R}$.*

The following lemma is crucial.

▶ **Lemma 34** (Free Points Reporting). *Let $\mathcal{R}$ be a set consisting of $\Theta(n)$ rectangles with corners in $[n]^2$. There is an algorithm that reports, for the input $\mathcal{R}$,*

- *all $\mathcal{R}$-free points in $O(n \log n + \mathsf{output} \cdot \log n)$ time, or*
- *for each $x \in [n]$, the $\mathcal{R}$-free point (x, y) with minimal y (if any) in $O(n \log n)$ time, or*
- *for an additional input $m \in [n]$, all $\mathcal{R}$-free points (x, y) with $x+y \le m$ in time $O(n \log n + \mathsf{output} \cdot \log n)$.*

Let A be a data structure storing an array A of n nonnegative integers and performing the following updates and queries. The update $\mathsf{Add}(y_1, y_2, \delta)$ adds the integer δ to all elements of $A[y_1..y_2]$ (δ can be negative if A is guaranteed to stay nonnegative at all times). The queries left0, all0, and $\mathsf{range0}(y_1, y_2)$ return, respectively, the index of the leftmost 0 in A, of all 0's in A, and of all 0's in $A[y_1..y_2]$; a query returns $\varnothing$ if the requested set of indices is empty. The proof of Lemma 34 is based on the following lemma.

▶ **Lemma 35.** *The data structure A can be implemented in $O(n)$ preprocessing time, $O(\log n)$ update time, and $O(\mathsf{output} \cdot \log n)$ query time, where* output *is the output size.*

Proof. We organize A as a lazy segment tree (see, e.g., [27]). The details are as follows.

We take a fully balanced binary tree with $2^{\lceil \log n \rceil}$ leaves and delete $2^{\lceil \log n \rceil} - n$ rightmost leaves together with all internal nodes having no leaves remained in their subtrees. The remaining leaves are enumerated from 1 to n in a natural order; every node is identified with the range of leaves in its subtree. Thus, leaf i represents $A[i]$ and node $[i..j]$ represents $A[i..j]$. Each node I stores links $I.\mathit{left}$ and $I.\mathit{right}$ to its children and also the numbers $I.\mathit{val}$ (value), $I.\mathit{min}$ (minimum), and $I.\mathit{mincnt}$ (counter), supporting the following invariants:

(i) for any leaf i, $A[i] = \sum_{I \ni i} I.\mathit{val}$;
(ii) for any node $I = [i..j]$, $\min\{A[y] \mid y \in I\} = I.\mathit{min} + \sum_{I \subset I'} I'.\mathit{val}$;
(iii) for any node $I = [i..j]$, $\min\{A[y] \mid y \in I\}$ occurs in $A[i..i]$ $I.\mathit{mincnt}$ times.

As a preprocessing, we assign each $A[i]$ to the corresponding leaf as both the value and the minimum, set values of internal nodes to 0, and compute their minima and counters in one bottom-up traversal. This procedure clearly takes $O(n)$ time.

The update $\mathsf{Add}(y_1, y_2, \delta)$ is performed by calling the recursive function $\mathsf{Update}(I, y_1, y_2, \delta)$ with $I = [1..n]$ (i. e., at the root of A). The tree of recursive calls contains at most four nodes per level of A. Therefore, the update takes $O(\log n)$ time.

```
- Update(I, y_1, y_2, δ):
    - If $I \cap [y_1..y_2] = \varnothing$: return $I.min$, $I.mincnt$
    - If $I \subseteq [y_1..y_2]$: add $\delta$ to both $I.val$ and $I.min$; return $I.min$, $I.mincnt$
    - Else: $min_1, mincnt_1 = \mathsf{Update}(I.left, y_1, y_2, \delta)$
            $min_2, mincnt_2 = \mathsf{Update}(I.right, y_1, y_2, \delta)$
            $I.min = I.val + \min\{min_1, min_2\}$
            $I.mincnt = mincnt_1 \cdot [min_1 \le min_2] + mincnt_2 \cdot [min_2 \le min_1]$
            return $I.min$, $I.mincnt$
```

Let us prove that $\mathsf{Update}(I, y_1, y_2, \delta)$ preserves all invariants. For (i) note that both sides remain the same if $i \notin [y_1..y_2]$ and get $+\delta$ if $i \in [y_1..y_2]$ (in the latter case, the value of exactly one node in the path from the root to i is changed). Similarly, both sides of (ii) remain the same if $I \cap [y_1..y_2] = \varnothing$ and increase by δ if $I \subseteq [y_1..y_2]$ (in the latter case, δ is added either to $I.min$ or to $.val$ of exactly one ancestor of I). In both cases (iii) holds as no changes were made. In particular, (ii) and (iii) hold for all leaves of A.

In the remaining case $\varnothing \subset I \cap [y_1..y_2] \subset I$ both (ii) and (iii) hold for the computed numbers $I.min$ and $I.mincnt$ if they hold for the minima and counters of the children of I. Hence, if an invariant is violated for I, it is also violated for a child of I. Since both invariants (ii) and (iii) hold for all leaves, they hold for all nodes.

It remains to describe queries. We say that I is a 0-node if $\min\{A[i] \mid i \in I\} = 0$. Obviously, all ancestors of a 0-node are 0-nodes, and at least one child of an internal 0-node is a 0-node. Note that $\min(A) = [1..n].min$. Thus, if $[1..n].min \ne 0$, the root is not a 0-node and so any query returns $\varnothing$. Otherwise, all or some 0-nodes that are leaves of A should be reported. To answer $\mathsf{all0}$, we perform a depth-first traversal of A, visiting only 0-nodes. For $\mathsf{range0}(y_1, y_2)$, we do a similar traversal but ignore all nodes representing ranges disjoint from $[y_1..y_2]$. Finally, to answer $\mathsf{left0}$ we just follow the leftmost path that consists of 0-nodes and connects the root to a leaf.

Observe that using (ii) one can decide in $O(1)$ time whether a child of a 0-node is a 0-node. Hence each of the queries spends $O(\log n)$ time per reported index, as required. ◀

Proof of Lemma 34. Let M be an $n \times n$ matrix such that $M[x, y]$ is the number of rectangles from $\mathcal{R}$ containing the point (x, y). Hence (x, y) is $\mathcal{R}$-free if and only if $M[x, y] = 0$. We report $\mathcal{R}$-free points in n phases, using the data structure A to implement a “sweeping line” strategy. At phase x, we compute the x'th row of M from its $(x-1)$'th row (the virtual “0th row” consists of all 0's) and then report the indices of 0's in the x'th row.

Let $R = [x_1..x_2] \times [y_1..y_2] \in \mathcal{R}$ be a rectangle. The points of the range $[y_1..y_2]$ are covered by R during the phases $x_1, x_1+1, \ldots, x_2$. According to this, we associate with R two updates of A: $\mathsf{Add}(y_1, y_2, 1)$ at phase x_1 and $\mathsf{Add}(y_1, y_2, -1)$ at phase $x_2 + 1$ (if $x_2 < n$). We group all updates by the phase they belong to and initialize A with an all-zero array A. In each phase, we perform all operations $\mathsf{Add}()$ of this phase and then query the indices of zeroes in the current array (the queries $\mathsf{all0}, \mathsf{left0}$, and $\mathsf{range0}(1, m-x)$ correspond to the first, second, and third statements of the lemma, respectively). As $|\mathcal{R}| = O(n)$, we perform, in total, $O(n)$ updates and $O(n)$ queries. Now the lemma follows from Lemma 35. ◀

To present a solution to the Shortest_2-cover problem, we need one more claim.

▷ Claim 36. If a point (x, y) is associated with a core 2-cover (X, Y), then $|X| + |Y| = x + y$ in the prefix-suffix case and $|X| + |Y| = x + y + 1 + b$ in the border-substring case with the border of length b.

Proof. In the prefix-suffix case, $X = S[1..x]$ and $Y = S[n - y + 1..n]$. In the border-substring case, $X = S[1..b]$ and $Y = S[f - x..f + y]$ for some position f. The claim follows. ◁

Proof of Theorem 2. The next algorithm solves Shortest_2-cover. We build all 2-dimensional 2CNF structures (Lemma 31). For the set $\mathcal{R}$ of each structure, we run the algorithm of Lemma 34 with the second option and choose the point (x, y) with the minimum sum of coordinates. From this pair we restore the associated 2-cover (X, Y). Due to Claim 36, (X, Y) has the minimum length among all 2-covers corresponding to this 2CNF structure. After processing all sets $\mathcal{R}$, we return the 2-cover of minimum length among those found.

Since each of $O(\log^2 n)$ sets $\mathcal{R}$ is computed in $O(n \log n)$ time (Lemma 31) and processed in $O(n \log n)$ time (Lemma 34), the time complexity is $O(n \log^3 n)$, as required. ◀

5.1 Report All 2-Covers Of Bounded Length

In this section, we sketch the proof of Theorem 1. As a preliminary step, the algorithm constructs the 2-cover oracle of Theorem 3. The first main step is similar to the proof of Theorem 2: the algorithm computes the set $\mathcal{C}_m$ of all core 2-covers of length at most m using the third statement of Lemma 34. The final step is to report all highly periodic 2-covers of length at most m by "extending" the core 2-covers.

For a string X, we define its *trimmed form* by setting $X_{\mathsf{trim}} = X$ if X is not highly periodic and $X_{\mathsf{trim}} = X[1..2\rho + |X| \bmod \rho]$ if X is highly ρ-periodic.

▷ Claim 37. If (X, Y) is a 2-cover of S and $|X| + |Y| \leq m$, then $(X_{\mathsf{trim}}, Y_{\mathsf{trim}}) \in \mathcal{C}_m$.

Proof. The pair $(X_{\mathsf{trim}}, Y_{\mathsf{trim}})$ is a 2-cover by Lemma 7, a core 2-cover by definitions of trimmed form and core 2-cover, and $|X_{\mathsf{trim}}| + |Y_{\mathsf{trim}}| \leq |X| + |Y| \leq m$. Hence $(X_{\mathsf{trim}}, Y_{\mathsf{trim}}) \in \mathcal{C}_m$. ◁

Consider the following process: for every 2-cover $(X, Y) \in \mathcal{C}_m$, report all its *associates*, which are 2-covers $(X', Y') \notin \mathcal{C}_m$ such that $X = X'_{\mathsf{trim}}$, $Y = Y'_{\mathsf{trim}}$, and $|X'| + |Y'| \leq m$. Claim 37 guarantees that the process reports all non-core 2-covers of length at most m. Note that a pair (X, Y) may have an associate only if X or Y is short periodic.

Let us describe how one pair is processed. Assume that $(X, Y) \in \mathcal{C}_m$ is a ps-cover. Let $X = S[1..2\rho + d]$ be short ρ-periodic and let Y be aperiodic. We initialize an iterator $q = 3$, and check if the following conditions hold:
(1) $q \cdot \rho + d + |Y| \leq m$ (in $O(1)$ time);
(2) $X_q = S[1..q \cdot \rho + d]$ is ρ-periodic (in $O(1)$ time by Lemma 16);
(3) the pair (X_q, Y) is a 2-cover (in one call to the oracle, $O(\log^3 n)$ time by Theorem 3).
If all three conditions are true, the algorithm reports (X_q, Y) as a 2-cover, assignes $q \leftarrow q + 1$ and checks the conditions again. Otherwise, the algorithm halts. It is easy to see that exactly all associates of (X, Y) are reported this way.

The case of aperiodic X and short periodic Y is symmetric to the above one. Now let X be short ρ-periodic and let Y be short τ-periodic; denote the suffix of S of length $|Y| + i\tau$ by Y_i. The algorithm first runs the above procedure for the pair (X, Y). Then it checks by Lemma 16 whether Y_1 is τ-periodic, and if yes, checks with the oracle whether (X, Y_1) is a 2-cover. If any of the conditions fail, the algorithm stops as Lemma 7 guarantees that the pair (X, Y) has no unreported associates. Otherwise, the algorithm runs the above procedure for the pair (X, Y_1), checks the conditions for Y_2, runs the above procedure for (X, Y_2) if the conditions hold, and so on.

The time complexity of the described algorithm is dominated by the queries to the oracle. We count separately T-queries and F-queries, which were answered by True and False, respectively. Consider processing the pair $(X, Y) \in \mathcal{C}_m$. Each T-query corresponds to a unique reported 2-cover. Each F-query ends an iteration of the outer loop (in the case where only one of X, Y is periodic, we count just one iteration). Every iteration, except possibly the last one, contains a T-query. Thus, we can charge at most two queries on each reported 2-cover, and charge the F-query from the last iteration on (X, Y). Therefore, the total number of queries over all pairs $(X, Y) \in \mathcal{C}_m$ is $O(\mathsf{output})$. By Theorem 3, the total running time is $O(\mathsf{output} \cdot \log^3 n)$. Adding the time for constructing the oracle, we get the bound required by Theorem 1.

The case of a bs-cover $(X, Y) \in \mathcal{C}_m$ is similar to the above one. We just point out a couple of important details. First, the border X can be a 1-cover. If X is aperiodic, then no processing is needed. But if X belongs to the group bor of ρ-periodic borders, we use the information obtained during the oracle construction: the length of the shortest border $U \in \mathsf{bor}$ that is not 1-cover, and the position f_{bor} not covered by U. We check with Lemma 16 whether Y covers f_{bor}, and if yes, run a reporting procedure, similar to the above, for the pair (U, Y). Second, when we extend a τ-periodic substring $Y = S[f_{\mathsf{bor}} - \ell..f_{\mathsf{bor}} + r]$ by a period, we can do it either to the right or to the left; respectively, we check whether any of the substrings $S[f_{\mathsf{bor}} - \ell - \tau..f_{\mathsf{bor}} + r]$, $S[f_{\mathsf{bor}} - \ell..f_{\mathsf{bor}} + r + \tau]$ is τ-periodic.

The detailed proof of Theorem 1 can be found in [8].

6 Conclusion and Future Work

In this paper we solved, in $\tilde{O}(n)$ time, a string problem with $\Theta(n^3)$ search space (this is the number of options to choose a prefix and a substring in a given string). A natural question for future study is how efficient our methods can be extended to compute λ-covers for $\lambda > 2$, where the search space is of size $\Theta(n^{2\lambda-1})$.

The algorithm of Radoszewski and Straszyński [26] for 2-covers with substrings of equal length smoothly generalizes to an $\tilde{O}(n^{\lambda-1})$-time algorithm computing the minimal λ-cover with the same length restriction. However, the search space in the restricted problem is only $\Theta(n^{\lambda})$. It is not clear if our approach can be extended to obtain a similar running time for general λ-covers. Indeed, each substring in the λ-cover needs its own “anchor” point f to be represented as $S[f - \ell..f + r]$. This adds 2 to the dimension of the problem even in the aperiodic case. While such an increase seems to be not a problem for the oracle construction, it heavily affects the free point reporting.

Another interesting challenge is to provide conditional lower bounds of the order $n^{\Omega(\lambda)}$ for λ-covers.

References

1 Ali Alatabbi, M. Sohel Rahman, and W. F. Smyth. Computing covers using prefix tables. *Discret. Appl. Math.*, 212:2–9, 2016. doi:10.1016/J.DAM.2015.05.019.

2 Amihood Amir, Avivit Levy, Moshe Lewenstein, Ronit Lubin, and Benny Porat. Can we recover the cover? *Algorithmica*, 81(7):2857–2875, 2019. doi:10.1007/S00453-019-00559-8.

3 Amihood Amir, Avivit Levy, Ronit Lubin, and Ely Porat. Approximate cover of strings. *Theor. Comput. Sci.*, 793:59–69, 2019. doi:10.1016/J.TCS.2019.05.020.

4 Alberto Apostolico and Andrzej Ehrenfeucht. Efficient detection of quasiperiodicities in strings. *Theoretical Computer Science*, 119(2):247–265, 1993. `doi:10.1016/0304-3975(93)90159-Q`.

5 Hideo Bannai, Tomohiro I, Shunsuke Inenaga, Yuto Nakashima, Masayuki Takeda, and Kazuya Tsuruta. The "runs" theorem. *SIAM J. Comput.*, 46(5):1501–1514, 2017. `doi:10.1137/15M1011032`.

6 Carl Barton, Tomasz Kociumaka, Chang Liu, Solon P. Pissis, and Jakub Radoszewski. Indexing weighted sequences: Neat and efficient. *Inf. Comput.*, 270, 2020. `doi:10.1016/J.IC.2019.104462`.

7 Omer Berkman, Costas S. Iliopoulos, and Kunsoo Park. The subtree max gap problem with application to parallel string covering. *Inf. Comput.*, 123(1):127–137, 1995. `doi:10.1006/INCO.1995.1162`.

8 Itai Boneh, Shay Golan, and Arseny M. Shur. String 2-covers with no length restrictions. *CoRR*, abs/2405.11475, 2024. `doi:10.48550/arXiv.2405.11475`.

9 Dany Breslauer. An on-line string superprimitivity test. *Inf. Process. Lett.*, 44(6):345–347, 1992. `doi:10.1016/0020-0190(92)90111-8`.

10 Bernard Chazelle. A functional approach to data structures and its use in multidimensional searching. *SIAM J. Comput.*, 17(3):427–462, 1988. `doi:10.1137/0217026`.

11 Richard Cole, CS Ilopoulos, Manal Mohamed, William F Smyth, and Lu Yang. The complexity of the minimum k-cover problem. *Journal of Automata, Languages and Combinatorics*, 10(5-6):641–653, 2005.

12 Patryk Czajka and Jakub Radoszewski. Experimental evaluation of algorithms for computing quasiperiods. *Theoretical Computer Science*, 854:17–29, 2021.

13 Jonas Ellert, Pawel Gawrychowski, and Garance Gourdel. Optimal square detection over general alphabets. In Nikhil Bansal and Viswanath Nagarajan, editors, *Proceedings of the 2023 ACM-SIAM Symposium on Discrete Algorithms, SODA 2023, Florence, Italy, January 22-25, 2023*, pages 5220–5242. SIAM, 2023. `doi:10.1137/1.9781611977554.CH189`.

14 Zvi Galil and Raffaele Giancarlo. Data structures and algorithms for approximate string matching. *Journal of Complexity*, 4(1):33–72, 1988. `doi:10.1016/0885-064X(88)90008-8`.

15 Pawel Gawrychowski, Jakub Radoszewski, and Tatiana Starikovskaya. Quasi-periodicity in streams. In Nadia Pisanti and Solon P. Pissis, editors, *30th Annual Symposium on Combinatorial Pattern Matching, CPM 2019, June 18-20, 2019, Pisa, Italy*, volume 128 of *LIPIcs*, pages 22:1–22:14. Schloss Dagstuhl – Leibniz-Zentrum für Informatik, 2019. `doi:10.4230/LIPICS.CPM.2019.22`.

16 Qing Guo, Hui Zhang, and Costas S Iliopoulos. Computing the λ-covers of a string. *Information Sciences*, 177(19):3957–3967, 2007.

17 Donald E. Knuth, James H. Morris Jr., and Vaughan R. Pratt. Fast pattern matching in strings. *SIAM J. Comput.*, 6(2):323–350, 1977.

18 Tomasz Kociumaka, Solon P. Pissis, Jakub Radoszewski, Wojciech Rytter, and Tomasz Walen. Fast algorithm for partial covers in words. *Algorithmica*, 73(1):217–233, 2015. `doi:10.1007/S00453-014-9915-3`.

19 Tomasz Kociumaka, Jakub Radoszewski, Wojciech Rytter, and Tomasz Walen. Optimal data structure for internal pattern matching queries in a text and applications. *CoRR*, abs/1311.6235, 2013. `arXiv:1311.6235`.

20 Gad M. Landau and Uzi Vishkin. Fast string matching with k differences. *Journal of Computer and System Sciences*, 37(1):63–78, 1988. `doi:10.1016/0022-0000(88)90045-1`.

21 Laurentius Leonard and Ken Tanaka. Suffix tree-based linear algorithms for multiple prefixes, single suffix counting and listing problems. *CoRR*, abs/2203.16908, 2022. `arXiv:2203.16908`.

22 Yin Li and William F. Smyth. Computing the cover array in linear time. *Algorithmica*, 32:95–106, 2002.

23 Dennis Moore and W.F. Smyth. An optimal algorithm to compute all the covers of a string. *Information Processing Letters*, 50(5):239–246, 1994. `doi:10.1016/0020-0190(94)00045-X`.

24 Dennis Moore and W.F. Smyth. A correction to "an optimal algorithm to compute all the covers of a string". *Information Processing Letters*, 54(2):101–103, 1995. doi:10.1016/0020-0190(94)00235-Q.

25 Alexandru Popa and Andrei Tanasescu. An output-sensitive algorithm for the minimization of 2-dimensional string covers. In T. V. Gopal and Junzo Watada, editors, *Theory and Applications of Models of Computation - 15th Annual Conference, TAMC 2019, Kitakyushu, Japan, April 13-16, 2019, Proceedings*, volume 11436 of *Lecture Notes in Computer Science*, pages 536–549. Springer, 2019. doi:10.1007/978-3-030-14812-6_33.

26 Jakub Radoszewski and Juliusz Straszyński. Efficient computation of 2-covers of a string. In *28th Annual European Symposium on Algorithms (ESA 2020)*. Schloss Dagstuhl – Leibniz-Zentrum für Informatik, 2020.

27 Mikhail Rubinchik and Arseny M. Shur. Counting palindromes in substrings. In Gabriele Fici, Marinella Sciortino, and Rossano Venturini, editors, *String Processing and Information Retrieval - 24th International Symposium, SPIRE 2017, Proceedings*, volume 10508 of *Lecture Notes in Computer Science*, pages 290–303. Springer, 2017.

28 Dan E. Willard. New data structures for orthogonal range queries. *SIAM J. Comput.*, 14(1):232–253, 1985. doi:10.1137/0214019.