Efficient Algorithms for Shortest Partial Seeds in Words

Tomasz Kociumaka[1⋆], Solon P. Pissis[2], Jakub Radoszewski[1], Wojciech Rytter[1,3], and Tomasz Waleń[1]

[1] Faculty of Mathematics, Informatics and Mechanics, University of Warsaw, Warsaw, Poland
[kociumaka,jrad,rytter,walen]@mimuw.edu.pl
[2] Department of Informatics, King's College London, London WC2R 2LS, UK
solon.pissis@kcl.ac.uk
[3] Faculty of Mathematics and Computer Science, Copernicus University, Toruń, Poland

Abstract. A factor u of a word w is a *cover* of w if every position in w lies within some occurrence of u in w. A factor u is a *seed* of w if it is a cover of a superstring of w. Covers and seeds extend the classical notions of periodicity. We introduce a new notion of *α-partial seed*, that is, a factor covering as a seed at least α positions in a given word. We use the Cover Suffix Tree, introduced recently in the context of *α-partial covers* (Kociumaka et al, CPM 2013); an $\mathcal{O}(n \log n)$-time algorithm constructing such a tree is known. However it appears that partial seeds are more complicated than partial covers—our algorithms require algebraic manipulations of special functions related to edges of the modified Cover Suffix Tree and the border array. We present an algorithm for computing shortest α-partial seeds that works in $\mathcal{O}(n)$ time if the Cover Suffix Tree is already given.

1 Introduction

Periodicity in words is a fundamental topic in combinatorics on words and string algorithms (see [5]). The concept of quasiperiodicity is a generalization of the notion of periodicity [1]. Quasiperiodicity enables detecting repetitive structure of words when it cannot be found using the classical characterizations of periods. Several types of quasiperiods have already been introduced. It depends on the type of quasiperiod what kinds of repetitive structure it allows to detect.

The best-known type of quasiperiodicity is the *cover* of word. A factor u of a word w is said to be a cover of w if every position in w lies within some occurrence of u in w, we also say that w is covered by u. An extension of the notion of cover is the notion of *seed*, in this case the positions covered by a seed

⋆ Supported by Polish budget funds for science in 2013-2017 as a research project under the 'Diamond Grant' program.

u are also positions within overhanging occurrences of u. More formally, u is a seed of w if w is a factor of a word y covered by u.

Several algorithms are known for computation of covers and seeds. A linear-time algorithm for computing the shortest cover of a word was proposed by Apostolico et al. [2], and a linear-time algorithm for computing all the covers was proposed by Moore & Smyth [12]. Linear-time algorithms providing yet more complete characterizations of covers by so-called cover arrays were given in [3, 11]. Seeds were first introduced by Iliopoulos, Moore, and Park [7] who presented an $\mathcal{O}(n \log n)$-time algorithm computing seeds. This result was improved recently by Kociumaka et al. [8] who gave a complex linear-time algorithm.

It remains unlikely that an arbitrary word has a cover or a seed shorter than the word itself. Due to this reason, relaxed variants of quasiperiodicity have been introduced. One of the ideas are *approximate covers* [13] and *approximate seeds* [4] that require each position to lie within an approximate occurrence of the corresponding quasiperiod. Another idea, introduced recently in [9], was the notion of *partial cover* that is required to cover a certain number of positions of a word. We extend the ideas of [9] and introduce the notion of *partial seed*.

Let $\mathcal{C}(u, w)$ denote the number of positions in w covered by (full) occurrences of the word u in w. The word u is called an *α-partial cover* of w if $\mathcal{C}(u, w) \geq \alpha$. We call a non-empty prefix of w that is also a suffix of u a *left-overhanging* occurrence of u in w. Symmetrically, a non-empty suffix of w which is a prefix of u is called a *right-overhanging* occurrence. Let $\mathcal{S}(u, w)$ denote the number of positions in w covered by full, left-overhanging, or right-overhanging occurrences of u in w. We call u an *α-partial seed* of w if $\mathcal{S}(u, w) \geq \alpha$. If the word w is clear from the context, we use the simpler notations of $\mathcal{C}(u)$ and $\mathcal{S}(u)$.

Example 1. If $w = \texttt{aaaabaabaaaaaba}$, see also Fig. 1, then

$$\mathcal{S}(\texttt{abaa}) = 12,\ \mathcal{S}(\texttt{aba}) = 10,\ \mathcal{S}(\texttt{ab}) = 7,\ \mathcal{S}(\texttt{a}) = 12.$$

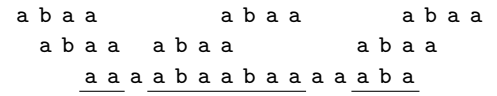

Fig. 1. The positions covered by `abaa` as a partial seed are underlined. The word `abaa` is a 12-partial seed of w, it has four overhanging occurrences and two full occurrences. Note that `a` is the shortest 12-partial seed of w.

We study the following two related problems.

PartialSeeds
Input: a word w of length n and a positive integer $\alpha \leq n$
Output: all shortest factors u of w such that $\mathcal{S}(u, w) \geq \alpha$

LimitedLengthPartialSeeds
Input: a word w of length n and an interval $[\ell, r]$, $0 < \ell \leq r \leq n$
Output: a factor u of w, $|u| \in [\ell, r]$, which maximizes $\mathcal{S}(u, w)$

In [9] a data structure called the *Cover Suffix Tree* and denoted by $CST(w)$ was introduced. For a word w of length n the size of $CST(w)$ is $\mathcal{O}(n)$ and the construction time is $\mathcal{O}(n \log n)$. In this article, we obtain the following results.

Theorem 2. *Given $CST(w)$, the* LimitedLengthPartialSeeds *problem can be solved in $\mathcal{O}(n)$ time.*

By applying binary search, Theorem 2 implies an $\mathcal{O}(n \log n)$-time solution to the PartialSeeds problem. However, this solution can be improved to an $\mathcal{O}(n)$-time algorithm, provided that $CST(w)$ is known.

Theorem 3. [Main result] *Given $CST(w)$, the* PartialSeeds *problem can be solved in $\mathcal{O}(n)$ time.*

Structure of the paper. In Section 2 we introduce basic notation related to words and suffix trees and recall the Cover Suffix Tree. Next in Section 3 we extend CST to obtain its counterpart suitable for computation of partial seeds, which we call the Seed Suffix Tree (SST). In Section 4 we introduce two abstract problems formulated in terms of simple functions which encapsulate the intrinsic difficulty of the two types of PartialSeeds problems. We present the solutions of the abstract problems in Section 5; this section is essentially the most involved part of our contribution. We summarize our results in the Conclusions section.

2 Preliminaries

Let us fix a word w of length n over a totally ordered alphabet Σ. For a factor v of w, by $Occ(v)$ we denote the set of positions where occurrences of v in w start. By $first(v)$ and $last(v)$ we denote $\min Occ(v)$ and $\max Occ(v)$, respectively.

By $w[i\,..\,j]$ we denote the factor starting at the position i and ending at the position j. Factors $w[1\,..\,i]$ are called *prefixes* of w, and factors $w[i\,..\,n]$ are called *suffixes* of w. Words shorter than w that are both prefixes and suffixes of w are called *borders* of w. By $\beta(w)$ we denote the length of the longest border of w. The border array $\beta[1\,..\,n]$ and reverse border array $\beta^R[1\,..\,n]$ of w are defined as follows: $\beta[i] = \beta(w[1\,..\,i])$ and $\beta^R[i] = \beta(w[i\,..\,n])$. The arrays β, β^R can be constructed in $\mathcal{O}(n)$ time [6].

The *suffix tree* of w, denoted by $ST(w)$, is the compacted suffix trie of w in which only branching nodes and suffixes are explicit. We identify the nodes of $ST(w)$ with the factors of w that they represent. An *augmented* suffix tree may contain some additional explicit nodes, called *extra* nodes. For an explicit node $v \neq \varepsilon$, we set $path(v) = (v_0, v_1, \ldots, v_k)$ where $v_0 = v$ and $v_1, \ldots, v_k$ are the implicit nodes on the path going upwards from v to its nearest explicit ancestor. E.g., in the right tree in Fig. 2 we have $path(v) = (v, v_1, v_2, v_3, v_4, v_5)$. We define the *locus* of a factor v' of w as a pair (v, j) such that $v' = v_j$ where $v_j \in path(v)$.

The Cover Suffix Tree is an augmented version of a suffix tree introduced in [9] that allows to efficiently compute $\mathcal{C}(v)$ for any explicit or implicit node, as shown in the following theorem.

Theorem 4 ([9]). *Let w be a word of length n. There exists an augmented suffix tree of size $\mathcal{O}(n)$, such that for each edge path(v) we have $\mathcal{C}(v_j) = c(v) - j\Delta(v)$ for some positive integers $c(v), \Delta(v)$. Such a tree together with values $c(v), \Delta(v)$, denoted as $CST(w)$, can be constructed in $\mathcal{O}(n \log n)$ time and $\mathcal{O}(n)$ space.*

Actually [9] provides explicit formulas for $c(v), \Delta(v)$ in terms of $Occ(v)$. Their form is not important here; the only property which we use is that $1 \leq \Delta(v) \leq |Occ(v)|$.

3 Seed Suffix Tree

CST introduces some extra nodes to ST thanks to which the cover index $\mathcal{C}(v_j)$ on each edge becomes a linear function: $\mathcal{C}(v_j) = c(v) - j\Delta(v)$. With seed index $\mathcal{S}(v_j)$, the situation is more complex. However, if we make some more nodes explicit, then $\mathcal{S}(v_j)$ becomes a relatively simple function. We call the resulting tree the *Seed Suffix Tree*, denoted by $SST(w)$.

Lemma 5. *Let w be a word of length n. We can construct an augmented suffix tree, denoted by $SST(w)$, of size $\mathcal{O}(n)$ such that for each node v there exists a function $\phi_v(x) = a_v x + b_v + \min(c_v, \beta[x])$ and a range $R_v = (\ell_v, r_v]$ such that for all $v_j \in$ path(v) we have $\mathcal{S}(v_j) = \phi_v(r_v - j)$. Additionally, $0 \leq a_v \leq |Occ(v)|$. The tree $SST(w)$, together with the border array β and tuples $(a_v, b_v, c_v, \ell_v, r_v)$ representing ϕ_v, can be constructed in $\mathcal{O}(n)$ time given $CST(w)$.*

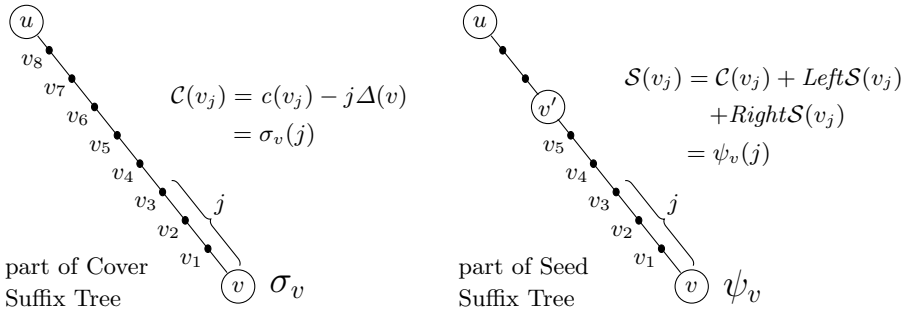

Fig. 2. In $CST(w)$ there is a constant-space description of a linear function σ_v associated with each explicit node v, which gives the values of $\mathcal{C}(v_j)$ for implicit nodes on the edge from v upwards. In $SST(w)$ there is a corresponding function ψ_v which is a combination of the linear function σ_v and two functions depending on border arrays. After suitable linear transformation of variable j, the function $\psi_v(j)$ is converted to a more convenient function $\phi_v(x)$. When transforming $CST(w)$ to $SST(w)$, some implicit nodes are made explicit to guarantee that ϕ_v has a simple form (v' on the figure).

Proof. For any factor v of w we define:

$$\mathit{Left}\mathcal{S}(v) = \min(\beta[\mathit{first}(v) + |v| - 1], \mathit{first}(v) - 1)$$
$$\mathit{Right}\mathcal{S}(v) = \min(\beta^R[\mathit{last}(v)], n - |v| + 1 - \mathit{last}(v)).$$

The following observation relates these values to $\mathcal{S}(v)$, see also Fig. 3.

Claim. $\mathcal{S}(v) = \mathcal{C}(v) + \mathit{Left}\mathcal{S}(v) + \mathit{Right}\mathcal{S}(v)$.

Proof (of the claim). $\mathcal{C}(v)$ counts all positions covered by full occurrences of v. We claim that the remaining positions covered by left-overhanging occurrences are counted by $\mathit{Left}\mathcal{S}(v)$. Let $p = \mathit{first}(v) + |v| - 1$. Note that $w[1\,..\,p]$ has v as a suffix, which means that $\beta[p]$ is the length of the longest left-overhanging occurrence of v. It covers $\beta[p]$ positions, but, among them, positions greater than or equal to $\mathit{first}(v)$ are already covered by a full occurrence of v. $\mathit{Right}\mathcal{S}(v)$ has a symmetric interpretation. □

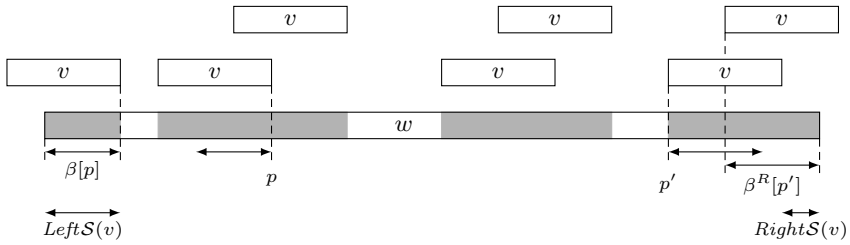

Fig. 3. The positions contained in $\mathcal{S}(v)$ are marked gray. In this case $\mathit{Left}\mathcal{S}(v) = \beta[\mathit{first}(v) + |v| - 1]$, and $\mathit{Right}\mathcal{S}(v) = n - |v| + 1 - \mathit{last}(v)$.

Consider an edge $\mathit{path}(v) = (v_0, \ldots, v_k)$. For any $0 \le j \le k$ we have:

$$\mathcal{C}(v_j) = c(v) - j\Delta(v)$$
$$\mathit{Left}\mathcal{S}(v_j) = \min(\beta[\mathit{first}(v) + |v| - j - 1], \mathit{first}(v) - 1)$$
$$\mathit{Right}\mathcal{S}(v_j) = \min(\beta^R[\mathit{last}(v)], n - |v| + j + 1 - \mathit{last}(v)).$$

We consider the function $\mathcal{S}(v_j) = \mathcal{C}(v_j) + \mathit{Left}\mathcal{S}(v_j) + \mathit{Right}\mathcal{S}(v_j)$. Note that only $\mathcal{C}(v_j)$ is a linear function of j. Also, $\mathit{Right}\mathcal{S}(v_j)$ is relatively simple. It either already is a linear function in the whole $\{0, \ldots, k\}$, or it becomes one in both parts of $\{0, \ldots, k\}$ if we split it at $j \in \{0, \ldots, k\}$ such that $\beta^R[\mathit{last}(v)] = n - |v| + j + 1 - \mathit{last}(v)$. We subdivide each $\mathit{path}(v)$ at v_j if such j exists. Note that we can easily update values c and Δ for newly created edges. Also, we make explicit at most $\mathcal{O}(n)$ nodes (at most one per edge of $\mathit{CST}(w)$), so the resulting tree $\mathit{SST}(w)$ has $\mathcal{O}(n)$ explicit nodes in total.

It remains to show that after these transformations $\mathcal{S}(v_j) = \phi_v(r_v - j)$ for $r_v = \mathit{first}(v) + |v| - 1$ and that the coefficients of ϕ_v can be efficiently computed.

We omit the explicit formulae in this version, they can be obtained with just a few simple algebraic transformations. The additional inequality $0 \leq a_v \leq |Occ(v)|$ follows from the property that $1 \leq \Delta(v) \leq |Occ(v)|$. □

The following observation is a direct consequence of Lemma 5.

Observation 6. *Given $SST(w)$ and a locus of u, a factor of w, one can compute $\mathcal{S}(u)$ in constant time.*

4 Reduction to two abstract problems

We say that an integer array $A[1\,..\,k]$ is a *linear-oscillation* array if

$$\sum_{i=1}^{k-1} |A[i] - A[i+1]| = \mathcal{O}(k).$$

Observation 7. *Any border array is a linear-oscillation array.*

To solve the LimitedLengthPartialSeeds problem, we make explicit all nodes corresponding to factors of length $\ell - 1$ and r. This way each edge either contains only nodes at tree levels in $\{\ell, \ldots, r\}$ or none of them. Note that the functions ϕ_v on the subdivided edges stay the same, only the ranges shrink. Consider the following abstract problem.

Problem A1
Input: a linear-oscillation array B of size n and m pairs (ϕ_i, R_i), where ϕ_i is a function $\phi_i(x) = a_i x + b_i + \min(c_i, B[x])$ and $R_i = (\ell_i, r_i] \subseteq [1, n]$ is a non-empty range
Output: the values $x_i = \operatorname{argmax}\{\phi_i(x) : x \in R_i\}$.[3]

Applying Problem A1 for $B = \beta$ and a query for each edge $path(v)$, we obtain $v_j \in path(v)$ maximizing $\mathcal{S}(v_j)$. Taking a global maximum over all edges containing factors of lengths within $\{\ell, \ldots, r\}$, we get the sought factor u, which maximizes $\mathcal{S}(u)$ among all factors of w with $|u| \in \{\ell, \ldots, r\}$. This results in the following lemma.

Lemma 8. *Given $SST(w)$ and an $\mathcal{O}(n+m)$-time off-line solution to Problem A1, the* LimitedLengthPartialSeeds *problem can be solved in $\mathcal{O}(n)$ time.*

To solve the PartialSeeds problem, we also apply Problem A1 to compute $\max \mathcal{S}(v_j)$ for each edge $path(v)$ of $SST(w)$ (this time we do not introduce any additional extra nodes). We say that an edge $path(v)$ is *feasible*, if $\max \mathcal{S}(v_j) \geq \alpha$, and *important* if it is feasible and no ancestor edge is feasible. It is easy to see

[3] For a set X and a function $f : X \to \mathbb{R}$, we define $\operatorname{argmax}\{f(x) : x \in X\}$ as the largest argument for which f attains its maximum value, that is, $\max\{x \in X : \forall_{x' \in X} f(x) \geq f(x')\}$ (we use the maximality of x later on). We assume $\max \emptyset = -\infty$ and $\min \emptyset = \infty$.

that all shortest α-partial seeds lie on important edges. Also, by Lemma 5, a_v summed over all feasible edges e_v is at most n. Consider the following abstract problem.

PROBLEM A2
Input: a linear-oscillation array B of size n, a positive integer α and m pairs (ϕ_i, R_i), where ϕ_i is a function $\phi_i(x) = a_i x + b_i + \min(c_i, B[x])$, $\sum a_i = \mathcal{O}(n)$, and $R_i = (\ell_i, r_i] \subseteq [1, n]$ is a range
Output: the values $\min\{x \in R_i : \phi_i(x) \geq \alpha\}$

Note that applied for $B = \beta$ and queries for all important edges, it gives the shortest α-partial seed within each important edge. Taking all globally shortest among these candidates, we get all shortest α-partial seeds. This results in the following lemma.

Lemma 9. *Given $SST(w)$ and $\mathcal{O}(n+m)$-time off-line solutions to Problems A1 and A2, the* PARTIALSEEDS *problem can be solved in $\mathcal{O}(n)$ time.*

5 Solutions to Abstract Problems

We begin with a few auxiliary definitions and lemmas. For a set $X \subseteq \mathbb{Z}$ and an integer $x \in \mathbb{Z}$ we define $\text{pred}(x, X) = \max\{y \in X : y \leq x\}$, $\text{succ}(x, X) = \min\{y \in X : y \geq x\}$, and $\text{rank}(x, X) = |\{y \in X : y \leq x\}|$. The technical proofs of the following two lemmas are left for the full version of the paper.

Lemma 10. *Let $Y[1\,..\,n]$ be an array of integers of magnitude $\mathcal{O}(n)$. For an integer k let $Y_{\geq k} = \{i : Y[i] \geq k\}$. Given m integer pairs (y_j, k_j) we can compute the values $\text{pred}(y_j, Y_{\geq k_j})$ and $\text{succ}(y_j, Y_{\geq k_j})$ in $\mathcal{O}(n + m)$ time.*

Lemma 11. *Assume we are given a family $\{X_a \subseteq [1\,..\,n] : a \in [1\,..\,n]\}$ of sets of total size $\mathcal{O}(n)$ and functions $\psi_a : [1\,..\,n] \to \mathbb{Z}$, each computable in constant time. Given m pairs (a_j, R_j) or m triplets (a_j, β_j, R_j), where $a_j \in [1\,..\,n]$, $R_j \subseteq [1\,..\,n]$ is an interval, and $\beta_j \in \mathbb{Z}$, we can compute*

$$\text{argmax}\{\psi_{a_j}(x) \ : \ x \in X_{a_j} \cap R_j\} \text{ in case of pairs, or}$$

$$\min\{x \in X_{a_j} \cap R_j \ : \ \psi_{a_j}(x) \geq \beta_j\} \text{ in case of triples}$$

offline in $\mathcal{O}(n + m)$ time.

The following simple result is the reason behind the linear-oscillation assumption in both problems.

Observation 12. *Let $F_a = \{x : B[x + 1] < B[x] - a\}$ for a linear-oscillation array B of size n. Then $\sum_{a=1}^{\infty} |F_a| = \mathcal{O}(n)$.*

Proof. Each $x \in \{1, \ldots, n-1\}$ belongs to F_a if and only if $a < B[x] - B[x+1]$. The sum is therefore bounded by the total decrease of B, which is $\mathcal{O}(n)$ for a linear-oscillation array. □

The following is a simplified version of Problem A1.

PROBLEM B1
Input: a linear-oscillation array B of size n and m pairs (ϕ_i, R_i), where ϕ_i is a function $\phi_i(x) = a_i x + B[x]$ and $R_i = (\ell_i, r_i] \subseteq [1, n]$ is a non-empty range
Output: the values $x_i = \text{argmax}\{\phi_i(x) : x \in R_i\}$

Lemma 13. *Problem A1 can be reduced to Problem B1 in $\mathcal{O}(n + m)$ time.*

Proof. Let

$$y_i = \max(\{x \in R_i : B[x] \geq c_i\} \cup \{\ell_i\}).$$

Consider any $x \in R_i$ such that $x \leq y_i$. If such x exists then $\phi_i(x) = a_i x + b_i + \min(c_i, B[x]) \leq a_i y_i + b_i + c_i = \phi_i(y_i)$. Consequently, $x_i = \text{argmax}\{\phi_i(x) : x \in R_i\} \geq y_i$. Note that for $x \in R_i$ such that $x > y_i$ we have $\phi_i(x) = a_i x + b_i + B[x]$. Thus, it suffices to solve Problem B1 for $R'_i = (y_i, r_i]$ (if $R'_i \neq \emptyset$). This way we find:

$$x'_i = \text{argmax}\{a_i x + B[x] : x \in R'_i\}.$$

Then x_i is guaranteed to be either x'_i or y_i, and it is easy to check in constant time which of the two it actually is.

The missing step of the reduction is determining y_i's. We claim that these values can be computed off-line in $\mathcal{O}(n + m)$ time. Instead of y_i it suffices to compute $y'_i = \max\{x \leq r_i : B[x] \geq c_i\}$ ($y'_i = -\infty$ if the set is empty). Then y_i can be determined as $\max(y'_i, \ell_i)$. Clearly $y'_i = \text{pred}(r_i, B_{\geq c_i})$, and these values can be computed in $\mathcal{O}(n + m)$ time using Lemma 10. □

Now, it remains to solve Problem B1. Recall sets F_a defined in Observation 12. Note that $x_i \in F_{a_i}$ or $x_i = r_i$. Indeed, if $x \notin F_{a_i}$ and $x \neq r_i$, then $x + 1 \in R_i$ and $\phi_i(x + 1) = a_i(x + 1) + B[x + 1] \geq a_i x + a_i + B[x] - a_i = \phi_i(x)$. Consequently, it is enough to compute

$$x'_i = \text{argmax}\{\phi_i(x) : x \in F_{a_i} \cap R_i\}$$

($x'_i = -\infty$ if the set is empty). Then x_i is either x'_i or r_i, and it is easy to check in constant time which of the two it actually is.

By Lemma 11 values x'_i can be computed in $\mathcal{O}(n + m)$ time. This concludes the solution to Problem B1, and together with Lemma 13 implies the following result.

Lemma 14. *Problem A1 can be solved in $\mathcal{O}(n + m)$ time.*

The following is a simplified version of Problem A2.

PROBLEM B2
Input: a linear-oscillation array B of size n and m triples (ϕ_i, α_i, R_i), where $\phi_i(x) = a_i x + B[x]$, $\sum a_i = \mathcal{O}(n)$, α_i is a positive integer and $R_i = (\ell_i, r_i] \subseteq [1, n]$ is a range
Output: the values $\min\{x \in R_i : \phi_i(x) \geq \alpha_i\}$

Lemma 15. *Problem A2 can be reduced to Problem B2 in $\mathcal{O}(n+m)$ time.*

Proof. We set $\alpha_i = \alpha - b_i$. Let $y_i = \left\lceil \frac{\alpha_i - c_i}{a_i} \right\rceil$. If $a_i = 0$, we set $y_i = \infty$ if $c_i < \alpha_i$ and $y_i = -\infty$ otherwise. Note that for $x \in R_i$ such that $x < y_i$ we have $a_i x + c_i < \alpha_i$, so $\phi_i(x) < \alpha$. Therefore $x_i \geq y_i$. On the other hand, if $x \geq y_i$ then $a_i x + c_i \geq \alpha_i$, so $\phi_i(x) \geq \alpha$ if and only if $a_i x + B[x] \geq \alpha_i$. Consequently, it suffices to solve Problem B2, with $R'_i = R_i \cap [y_i, \infty)$. □

It remains to solve Problem B2. Recall sets F_a from Observation 12 and set

$$G_a = F_a \cup \{x \in [1\,..\,n] \ : \ x \bmod (1 + a^2) = 0\}.$$

It holds that

$$\sum_{a=0}^{\infty} |G_a| \leq \sum_{a=0}^{\infty} |F_a| + n \sum_{a=0}^{\infty} \tfrac{1}{1+a^2} = \mathcal{O}(n)$$

by Observation 12 and since $\sum_{a=0}^{\infty} \frac{1}{1+a^2}$ is $\mathcal{O}(1)$. Note that Lemma 10 applied for an array Y with $Y[x] = B[x] - B[x+1] - 1$ lets us obtain $\mathrm{pred}(x, F_a)$ and $\mathrm{succ}(x, F_a)$. With simple arithmetics we can use these values to compute $\mathrm{pred}(x, G_a)$ and $\mathrm{succ}(x, G_a)$. Assume that x_i exists. Let

$$x'_i = \min(r_i, \mathrm{succ}(x_i, G_{a_i})), \quad x''_i = \max(\ell_i, \mathrm{pred}(x_i - 1, G_{a_i})).$$

Observe ϕ_i is non-decreasing within $R'_i = (x''_i, x'_i]$. Indeed, if $x, x+1 \in R'_i$, then $x \notin F_{a_i}$, so $\phi_i(x+1) \geq \phi_i(x)$, as noted in the solution to Problem B1. We claim that R'_i can be computed in $\mathcal{O}(n+m)$ time. Since $x''_i = \max(\ell_i, \mathrm{pred}(x'_i - 1, G_{a_i}))$ we can compute x''_i once we have x'_i. Thus, the main challenge is to compute x'_i. By monotonicity of ϕ_i in R'_i, we conclude that $\phi_i(x'_i) \geq \alpha_i$. On the other hand, for any $x \in R_i$ such that $x < x_i$ we have $\phi_i(x) < \alpha_i$. Moreover any $x \in R_i \cap G_{a_i}$ smaller than x'_i is smaller than x_i, so $x'_i = \min(\{x \in R_i \cap G_{a_i} : \phi_i(x) \geq \alpha_i\} \cup \{r_i\})$, and such values can be computed off-line in $\mathcal{O}(n+m)$ time by Lemma 11.

Once we have the interval R'_i, by monotonicity of ϕ_i on R'_i, we can find x_i using binary search in $\mathcal{O}(\log a_i)$ time, because $|R'_i| \leq a_i^2 + 1$. The total time complexity is $\mathcal{O}(n + m + \sum_i \log a_i)$, and since $\sum_i a_i = \mathcal{O}(n)$ in Problem B2, this reduces to $\mathcal{O}(n+m)$, which implies the following result.

Lemma 16. *Problem A2 can be solved in $\mathcal{O}(n+m)$ time.*

6 Conclusions

We are now ready to combine all the results obtained so far.

Theorem 2. *Given $CST(w)$, the* LimitedLengthPartialSeeds *problem can be solved in $\mathcal{O}(n)$ time.*

Proof. First, we apply Lemma 5 and construct $SST(w)$. Then, we solve the LimitedLengthPartialSeeds problem using Lemma 8, plugging the algorithm of Lemma 14 for Problem A1. □

Theorem 3. *Given $CST(w)$, the* PartialSeeds *problem can be solved in $\mathcal{O}(n)$ time.*

Proof. We proceed much as in Theorem 2: we construct $SST(w)$ and solve PartialSeeds problem using Lemma 9, plugging the algorithms of Lemmas 14 and 16 for Problems A1 and A2, respectively. □

An interesting open question is whether one can compute the shortest α-partial seed for each $\alpha \in \{1, \ldots, n\}$ any faster than applying n times Theorem 3. The corresponding problem for partial covers is known to have an $\mathcal{O}(n \log n)$-time solution [10].

References

1. A. Apostolico and A. Ehrenfeucht. Efficient detection of quasiperiodicities in strings. *Theor. Comput. Sci.*, 119(2):247–265, 1993.
2. A. Apostolico, M. Farach, and C. S. Iliopoulos. Optimal superprimitivity testing for strings. *Inf. Process. Lett.*, 39(1):17–20, 1991.
3. D. Breslauer. An on-line string superprimitivity test. *Inf. Process. Lett.*, 44(6):345–347, 1992.
4. M. Christodoulakis, C. S. Iliopoulos, K. Park, and J. S. Sim. Approximate seeds of strings. *Journal of Automata, Languages and Combinatorics*, 10(5/6):609–626, 2005.
5. M. Crochemore, L. Ilie, and W. Rytter. Repetitions in strings: Algorithms and combinatorics. *Theor. Comput. Sci.*, 410(50):5227–5235, 2009.
6. M. Crochemore and W. Rytter. *Jewels of Stringology.* World Scientific, 2003.
7. C. S. Iliopoulos, D. W. G. Moore, and K. Park. Covering a string. *Algorithmica*, 16(3):288–297, 1996.
8. T. Kociumaka, M. Kubica, J. Radoszewski, W. Rytter, and T. Waleń. A linear time algorithm for seeds computation. In Y. Rabani, editor, *SODA*, pages 1095–1112. SIAM, 2012.
9. T. Kociumaka, S. P. Pissis, J. Radoszewski, W. Rytter, and T. Waleń. Fast algorithm for partial covers in words. In J. Fischer and P. Sanders, editors, *CPM*, volume 7922 of *Lecture Notes in Computer Science*, pages 177–188. Springer, 2013.
10. T. Kociumaka, S. P. Pissis, J. Radoszewski, W. Rytter, and T. Waleń. Fast algorithm for partial covers in words. *ArXiv e-prints, arXiv:1401.0163 [cs.DS]*, Dec. 2013.
11. Y. Li and W. F. Smyth. Computing the cover array in linear time. *Algorithmica*, 32(1):95–106, 2002.
12. D. Moore and W. F. Smyth. An optimal algorithm to compute all the covers of a string. *Inf. Process. Lett.*, 50(5):239–246, 1994.
13. J. S. Sim, K. Park, S. Kim, and J. Lee. Finding approximate covers of strings. *Journal of Korea Information Science Society*, 29(1):16–21, 2002.