A Linear-Time Algorithm for Seeds Computation

TOMASZ KOCIUMAKA, University of Warsaw and Bar-Ilan University
MARCIN KUBICA, JAKUB RADOSZEWSKI, WOJCIECH RYTTER, and
TOMASZ WALEŃ, University of Warsaw

A seed in a word is a relaxed version of a period in which the occurrences of the repeating subword may overlap. Our first contribution is a linear-time algorithm computing a linear-size representation of all seeds of a word (the number of seeds might be quadratic). In particular, one can easily derive the shortest seed and the number of seeds from our representation. Thus, we solve an open problem stated in a survey by Smyth from 2000 and improve upon a previous $O(n \log n)$-time algorithm by Iliopoulos et al. from 1996. Our approach is based on combinatorial relations between seeds and subword complexity (used here for the first time in the context of seeds).

In previous papers, compact representations of seeds consisted of two independent parts operating on the suffix tree of the input word and the suffix tree of its reverse, respectively. Our second contribution is a novel and significantly simpler representation of all seeds that avoids dealing with the suffix tree of the reversed word. This result is also of independent interest from a combinatorial point of view.

A preliminary version of this work, with a much more complex algorithm constructing a representation of seeds on two suffix trees, was presented at the 23rd Annual ACM-SIAM Symposium on Discrete Algorithms (SODA'12).

CCS Concepts: • **Theory of computation → Pattern matching**;

Additional Key Words and Phrases: Seed of a word, quasiperiodicity, suffix tree, subword complexity

ACM Reference format:
Tomasz Kociumaka, Marcin Kubica, Jakub Radoszewski, Wojciech Rytter, and Tomasz Waleń. 2020. A Linear-Time Algorithm for Seeds Computation. *ACM Trans. Algorithms* 16, 2, Article 27 (April 2020), 23 pages.
https://doi.org/10.1145/3386369

1 INTRODUCTION

The notion of periodicity in words is widely used in many fields, such as combinatorics on words, pattern matching, data compression, automata theory, formal language theory, and molecular biology (see [44]). The concept of quasiperiodicity, introduced by Apostolico and Ehrenfeucht [7],

T. Kociumaka was supported by ISF grants (1278/16, 824/17, and 1926/19), a BSF grant (2018364), and an ERC grant MPM (683064) under the EU's Horizon 2020 Research and Innovation Programme. J. Radoszewski was supported by the Foundation for Polish Science project "Algorithms for text processing with errors and uncertainties" (POIR.04.04.00-00-24BA/16), co-financed by the European Union under the European Regional Development Fund.
Authors' addresses: T. Kociumaka, Institute of Informatics, University of Warsaw, Banacha 2, Warsaw, 02-097, Poland, and Department of Computer Science, Bar-Ilan University, Ramat-Gan, 5290002, Israel; email: kociumaka@mimuw.edu.pl; M. Kubica, J. Radoszewski, W. Rytter, and T. Waleń, Institute of Informatics, University of Warsaw, Banacha 2, Warsaw, 02-097, Poland; emails: {kubica, jrad, rytter, walen}@mimuw.edu.pl.
Permission to make digital or hard copies of all or part of this work for personal or classroom use is granted without fee provided that copies are not made or distributed for profit or commercial advantage and that copies bear this notice and the full citation on the first page. Copyrights for components of this work owned by others than ACM must be honored. Abstracting with credit is permitted. To copy otherwise, or republish, to post on servers or to redistribute to lists, requires prior specific permission and/or a fee. Request permissions from permissions@acm.org.
© 2020 Association for Computing Machinery.
1549-6325/2020/04-ART27 $15.00
https://doi.org/10.1145/3386369

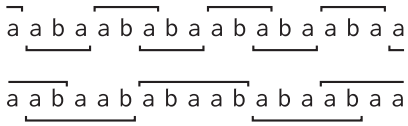

Fig. 1. The word aba (above) is the shortest seed of the word w = aabaababaabababaabaa. Another seed of w is abaab (below). Two of its "overhangs" correspond to boundary subwords aab and abaa. In total, the word w has 35 distinct seeds, but it does not have a non-trivial cover.

is a generalization of the notion of periodicity: A quasiperiodic word is entirely covered by occurrences of another (shorter) word, called the *quasiperiod* or the *cover*. The occurrences of the quasiperiod may overlap, whereas in a periodic repetition the occurrences of the period do not overlap. Hence, quasiperiodicity enables detection of repetitive structure of words that cannot be found using the classic characterizations in terms of periods.

An extension of the notion of a cover is the notion of a *seed*: a cover that is not necessarily aligned with the ends of the word being covered but is allowed to overflow on either side (see Figure 1). More formally, a word v is a *seed* of w if v is a subword of w and w is a subword of some word u covered by v.

Previous results. Seeds were first introduced and studied by Iliopoulos et al. [33], who gave an $O(n \log n)$-time algorithm computing a linear representation of all seeds of a given word. For the next 15 years, no $o(n \log n)$-time algorithm was known for this problem. Smyth [51] formulated computing all seeds of a word in linear time as an open problem in his survey. Berkman et al. [10] gave a parallel algorithm computing all seeds in $O(\log n)$ time and $O(n^{1+\varepsilon})$ space (for any positive ε) using n processors in the CRCW PRAM model. Much later, Christou et al. [15] proposed an alternative sequential $O(n \log n)$-time algorithm for computing the shortest seed.

In contrast, a linear-time algorithm finding the shortest cover of a word was given by Apostolico et al. [8] and later on improved into an online algorithm by Breslauer [12]. Moore and Smyth [46, 47] proposed a linear-time algorithm computing all covers of a word, whereas Li and Smyth [42] afterward developed an online algorithm for the problem of representing all covers of all prefixes of a word.

Both covers and seeds have also been extended in the sense that several subwords are considered instead of a single subword [35]. This way, the notions of k-covers [16, 32], λ-covers [28], and λ-seeds [26] were introduced. Notions of approximate quasiperiodicity, including approximate covers and seeds [4–6, 14, 30, 50], partial covers and seeds [23, 29, 39, 40], and approximate λ-covers [27], were also studied.

Another line of research is finding maximal quasiperiodic subwords of a word. This notion resembles maximal repetitions (runs) in a word [41]. Two $O(n \log n)$-time algorithms for reporting all maximal quasiperiodic subwords of a word of length n have been proposed by Brodal and Pedersen [13] and Iliopoulos and Mouchard [34]; these results improved upon the initial $O(n \log^2 n)$-time algorithm by Apostolico and Ehrenfeucht [7].

Our result. We present a linear-time algorithm computing the set $\mathrm{Seeds}(w)$ of all seeds of a given word w. As illustrated in the following Example 1.1, the number of seeds can be quadratic in the length $|w|$ (contrary to the number of covers, which is always linear). Consequently, our algorithm returns a linear-size *package representation* of the set $\mathrm{Seeds}(w)$, which allows finding a shortest seed and the number of all seeds in a very simple way.

Our procedure assumes that the alphabet Σ of the input word w consists of integers from the set $\{0, \ldots, n^{O(1)}\}$, where n is the length of the word w.

Example 1.1. The following word of length $4m + 3$ contains $\Theta(m^2)$ different seeds:

$$w = \mathsf{a}^m\mathsf{b}\mathsf{a}^m\mathsf{b}\mathsf{a}^m\mathsf{b}\mathsf{a}^m.$$

Those seeds are $\mathsf{a}^i\mathsf{b}\mathsf{a}^j$ with $i + j \geq m$ and $0 \leq i, j \leq m$.

Package representation of seeds. We assume that the letters of a word w are numbered from 1 to $|w|$ (i.e., $w[1], \ldots, w[|w|]$). By $w[i\,..\,j]$, we denote the subword $w[i] \cdots w[j]$. We introduce packages that are collections of consecutive prefixes of a subword of w. More formally, for positive integers $i \leq j_1 \leq j_2 \leq |w|$, we define a *package*:

$$\mathsf{pack}(i, j_1, j_2) = \{w[i\,..\,j] : j_1 \leq j \leq j_2\}.$$

If $\mathcal{L}$ is a set of ordered integer triples, then we denote

$$\mathsf{PACK}(\mathcal{L}) = \bigcup_{(i,j_1,j_2)\in\mathcal{L}} \mathsf{pack}(i, j_1, j_2).$$

The output of our algorithm, called the *package representation* of the set $\mathsf{Seeds}(w)$, consists of a set $\mathcal{L}$ of ordered integer triples such that $\mathsf{Seeds}(w) = \mathsf{PACK}(\mathcal{L})$ and all packages in the representation are pairwise disjoint. (In other words, each seed belongs to exactly one package.)

Let us recall that a *suffix trie* of w is the trie of all suffixes of w and a *suffix tree* of w is a compacted version of the suffix trie, in which nodes with exactly one child become implicit (possibly except for the root). Every package can be represented as a path in the suffix trie of w, and hence, as a path in the suffix tree of w that connects two nodes, possibly implicit.

Example 1.2. For a word $w = \mathsf{ababaabaab}$, a package representation of the set $\mathsf{Seeds}(w)$ is

$$\mathcal{L} = \{(1, 3, 3), (2, 9, 10), (1, 8, 10), (3, 10, 10), (3, 7, 8), (4, 8, 8)\}.$$

It corresponds to the following set $\mathsf{PACK}(\mathcal{L})$ of all seeds of w:

		ababaaba			
	babaabaa	ababaabaa		abaab	
aba	babaabaab	ababaabaab	abaabaab	abaaba	baaba
pack(1, 3, 3)	pack(2, 9, 10)	pack(1, 8, 10)	pack(3, 10, 10)	pack(3, 7, 8)	pack(4, 8, 8)

This collection of packages is also illustrated in Figure 2 as a set of disjoint paths in the suffix trie of w.

Previous compact representation of seeds. The original linear-size representation of seeds by Iliopoulos et al. [33], which was also employed in the preliminary version of our work [37], requires partitioning the set $\mathsf{Seeds}(w)$ into two disjoint subsets: Type-A seeds are represented as paths in the suffix trie of w, whereas (the reversals of) type-B seeds admit a similar representation in the suffix trie of the reversed word w^R. For both types, the number of reported paths is shown to be linear using a simple argument that each path can be uniquely extended to an edge of the corresponding suffix tree.

Remark 1. Example 1.2 illustrated in Figure 2 shows that the set of all seeds (rather than just type-A seeds) does not satisfy this property. Indeed, abaaba and abaabaab are both seeds of $w = \mathsf{ababaabaab}$ that occur in the same edge of the suffix tree, yet they cannot be reported on a single path because abaabaa is not a seed of w.

The *subword complexity* of a word w is a function that gives, for every $\ell = 1, \ldots, n$, the number of different subwords of length ℓ. For example, for $w = \mathsf{ababaabaab}$ and $\ell = 3$, there are four different length-3 subwords of w (i.e., aab, aba, baa, bab). Subword complexity is a well-studied notion in

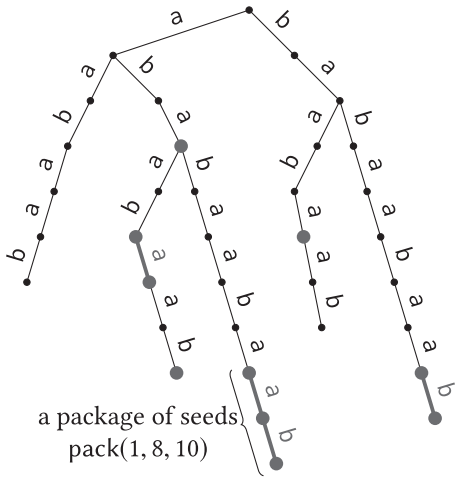

Fig. 2. The packages from Example 1.2 illustrated as paths (in bold) in the uncompressed suffix trie of $w =$ ababaabaab.

combinatorics on words (e.g., see Lothaire [43]), and it also has connections with compression [31, 38, 49].

Structure of the article. We start with a preliminary Section 2, where we recall several classic notions, relate packages to suffix trees, and provide equivalent formal definitions of seeds. Next, in Section 3, we prove that seeds admit a package representation of linear size. Some of the underlying arguments are stated in an algorithmic way so that they can be used in the subsequent Section 4 as subroutines of a procedure efficiently computing seeds of length $\Theta(n)$. In Section 5, we provide the novel relation between seeds and subword complexity, which is the key combinatorial contribution behind our main recursive algorithm described in Section 6. The implementation of an auxiliary operation on package representations (applied to merge the results of recursive calls) is deferred to Section 8. It uses a linear-time offline procedure answering weighted ancestor queries in a weighted tree, which we develop in Section 7, that is also of independent interest.

Our techniques. Our linear-time solution relies on several combinatorial and algorithmic tools:

— *Compact representation of seeds*: Despite its quadratic size and the failure of the naive argument (Remark 1), the set $\mathrm{Seeds}(w)$ always admits a package representation of linear size (Section 3). Although this fact is not essential for our algorithm (see [37]), it makes our results much simpler and cleaner.
— *Combinatorial properties of seeds*: The connection between seeds and subword complexity gives an efficient reduction to a set of recursive calls of total size decreased by a constant factor (see Sections 5 and 6).
— *Interpretation of packages as paths on the suffix trie*: Packages naturally correspond to paths in the suffix trie, which can be easily stored using the suffix tree. We use this interpretation both to derive the combinatorial upper bound on the package representation size (Section 3) and in the algorithmic construction of the package representation of long seeds (Section 4). The new linear-time offline algorithm answering weighted ancestor queries (Section 7) lets us efficiently map packages on the suffix tree.
— *Efficient manipulation of package representations*: Package representations provide a convenient way of interpreting the results of recursive calls (seeds of certain subwords) as

families of subwords of the whole word w. This allows for a simple linear-time procedure aggregating the results of the recursive calls (Section 8).

2 PRELIMINARIES

Let w be a *word* (a *string*) over an alphabet $\Sigma \subseteq \{0, \ldots, n^{O(1)}\}$, where n is the length of the word, also denoted as $|w|$. By $w[i]$, for $1 \leq i \leq |w|$, we denote the ith letter of w. By $Alph(w)$, we denote the set of letters occurring in w. A word u is a *subword* (or a *substring*) of w if $u = w[i] \cdots w[j]$ for some $1 \leq i \leq j \leq |w|$. We then say that u occurs in w at position i, and we denote by $Occ(u, w)$ (or $Occ(u)$ if w is clear from the context) the set of positions where u occurs in w. We denote the set of length-ℓ subwords of w by $\mathrm{SUB}_\ell(w)$. The *subword complexity* of a word w is a function that gives the number of subwords of a given length ℓ (i.e., $|\mathrm{SUB}_\ell(w)|$). If u is a subword of w, then we also say that w is a *superstring*[1] of u.

For $1 \leq i \leq j \leq |w|$, we denote by $w[i\,..\,j]$ the occurrence of $w[i] \cdots w[j]$ in w at position i. We call $w[i\,..\,j]$ a *fragment* of w; formally, a fragment can be interpreted as a triple (w, i, j) consisting of the word w and the positions i, j. We say that two fragments (perhaps in different words) *match* if they are occurrences of the same word. The length of the fragment $w[i\,..\,j]$ is defined as $|w[i\,..\,j]| = j - i + 1$. Note that a word w of length n has exactly $n - \ell + 1$ length-ℓ fragments. However, some of these fragments may match so that $|\mathrm{SUB}_\ell(w)| < n - \ell + 1$ is possible.

A fragment of w other than the whole word w is called *proper*. A fragment starting at position 1 is called a *prefix* of w, and a fragment ending at position $|w|$ is called a *suffix* of w.

A *border* of w is a subword of w that occurs both as a prefix and as a suffix of w. An integer p, $1 \leq p \leq n$, is a *period* of a word w if $w[i] = w[i + p]$ for $1 \leq i \leq |w| - p$. It is well known that p is a period of w if and only if w has a border of length $|w| - p$ (see [17, 18]). Moreover, the periodicity lemma of Fine and Wilf [22] asserts that if a word w has periods p and q such that $p + q - \gcd(p, q) \leq |w|$, then w also has a period $\gcd(p, q)$.

Throughout the article, by w we denote the word whose seeds are to be computed and by n its length.

2.1 Tries, Suffix Trees, and Package Representations

A *trie* is a rooted tree whose nodes correspond to prefixes of words in a given (finite) family W. If v is a node, then the corresponding prefix v is called the *value* of the node. The node with value v is called the *locus* of v. The parent-child relation in the trie is defined so that the root is the locus of the empty word, whereas the parent μ of a node v is the locus of the value of v with the last character removed. This character is the *label* of the edge from μ to v. In general, if μ is an ancestor of v, then the label of the path from μ to v is the concatenation of edge labels on the path. A trie of the set W containing all suffixes of a word babaad is shown in Figure 3.

A node is *branching* if it has at least two children and *terminal* if its value belongs to W. A *compacted trie* is obtained from the underlying trie by dissolving all nodes except for the root, branching nodes, and terminal nodes. In other words, we compress paths of non-terminal nodes with single children, and thus the number of remaining nodes becomes bounded by $2|W|$. We refer to all preserved nodes of the trie as *explicit* (since they are stored explicitly) and to the dissolved ones as *implicit*. If v is the locus of a word v in an uncompacted trie, then the locus of v in the corresponding compacted trie is defined as (μ, d), where μ is the lowest explicit descendant of v, and d is the distance (in the uncompacted trie) from v to μ. Note that $\mu = v$ and $d = 0$ if v is

[1]Notice that the term *superword* would be consistent with our notation, but it is not well established, so we decided not to use it. At the same time, *substring complexity* is a very rarely used synonym of *subword complexity*.

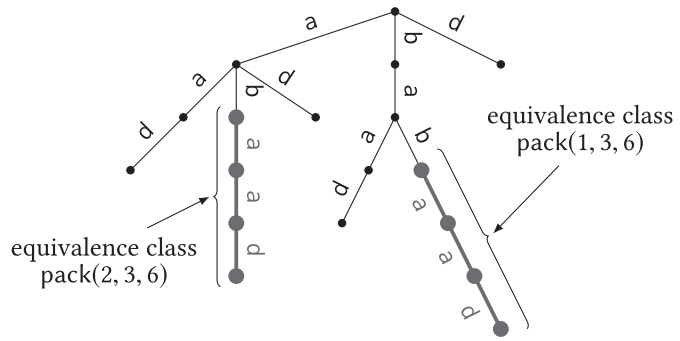

Fig. 3. The suffix trie of the word babaad. The equivalence classes correspond to compacted edges (excluding their topmost nodes). Two of them are marked in the figure: {ab, aba, abaa, abaad} = pack(2, 3, 6) and {bab, baba, babaa, babaad} = pack(1, 3, 6).

explicit. Edges of a compacted trie correspond to paths in the underlying trie, and thus their labels are non-empty words, typically stored as references to fragments of the words in W.

The *suffix trie* of word w is the trie of all suffixes of w (see Figure 3), with the locus $w[i\,..\,n]$ labeled by the position i. Consequently, there is a natural bijection between subwords of w and nodes of the suffix trie; we often use it to identify subwords of w with their loci in the suffix trie.

The *suffix tree* of w [52] is the compacted suffix trie of w. For a word of length n, it takes $O(n)$ space and can be constructed in $O(n)$ time either directly [21] or from the suffix array of w (see [17, 18, 36, 45]).

We say that two subwords u and v of w are *equivalent*, which we denote as $u \approx v$, if $Occ(u, w) = Occ(v, w)$. The equivalence classes of this relation correspond to edges of the suffix tree of w, as shown in the following observation and Figure 3 (see [17]).

Observation 2. *Each equivalence class of $\approx$ is of the form $E = \mathrm{pack}(i, j_1, j_2)$ for some positions $i \le j_1 \le j_2$ and corresponds to the set of all nodes on an edge of the suffix tree of w (excluding the topmost explicit node). Hence, there are at most $2|w|$ equivalence classes.*

In Section 8, we heavily use the connections between suffix trees and package representation to develop the following auxiliary procedure:

Combine$(R_1, \ldots, R_k)$: Given package representations of sets $R_1, \ldots, R_k$ of subwords of a word w, compute a smallest package representation of $\bigcap_{i=1}^{k} R_i$.

Note that the Combine operation is much more complicated than a simple intersection of sets of intervals. The following lemma is proved in Section 8.

Lemma 2.1. *For a word w of length n and sets $R_1, \ldots, R_k$ of subwords of w, given in package representations of total size N, Combine$(R_1, \ldots, R_k)$ can be implemented in $O(n + N)$ time. The size of the resulting package representation is at most N.*

2.2 Seeds: Formal Definition and Equivalent Characterizations

We say that a fragment $w[i\,..\,j]$ *covers* a position k (or that the position k lies within $w[i\,..\,j]$) if $i \le k \le j$. A word v is a *cover* of word w if the occurrences of v cover all positions in w. A word v is a *seed* of word w if it is a subword of w and a cover of a superstring of w. This definition immediately implies the following observation.

Observation 3. *Let v be a seed of word w. If v occurs in a subword w' of w, then v is a seed of w'.*

We denote by $\mathrm{Seeds}_I(w)$ the set of all seeds of w with lengths in the interval I. We also denote $\mathrm{Seeds}(w) = \mathrm{Seeds}_{[1..n]}(w)$, $\mathrm{Seeds}_{\le k}(w) = \mathrm{Seeds}_{[1..k]}(w)$, $\mathrm{Seeds}_{\ge k}(w) = \mathrm{Seeds}_{[k..n]}(w)$, and $\mathrm{Seeds}_k(w) = \mathrm{Seeds}_{[k..k]}(w)$.

A *left-overhang* of v in w is a prefix of w that matches a proper suffix of v. Symmetrically, a *right-overhang* of v in w is a suffix of w that matches a proper prefix of v. The length of the overhang is the length of the corresponding prefix/suffix of w. In this context, usual occurrences are sometimes called *full*, whereas a *generalized* occurrence is a full occurrence or an overhang. A known more operational definition of seeds can be formulated in terms of generalized occurrences as follows (e.g., see [39] and Figure 1).

Fact 4 (Alternative definition of seeds). *A subword v of word w is a seed of w if and only if each position in w is covered by a full occurrence, a left-overhang, or a right-overhang of v in w.*

Recall that $\mathrm{SUB}_k(w)$ denotes the set of all length-k subwords of w. Fact 4 lets us show that the family $\mathrm{SUB}_{2\ell-1}(w)$ of subwords of length $2\ell-1$ of a word w uniquely determines the length-ℓ seeds of w.

Lemma 2.2. *Let v be a non-empty subword of w such that $2|v|-1 \le n$. The following conditions are equivalent:*

(1) v is a seed of w;
(2) v is a seed of every subword of w of length $2|v|-1$.

Proof. (1) $\Rightarrow$ (2). Consider $s \in \mathrm{SUB}_{2|v|-1}(w)$ and observe that it occurs as $w[i-|v|+1\,..\,i+|v|-1]$ for some position i, $|v| \le i \le n-|v|+1$. The position i can only be covered by a full occurrence of v contained within $w[i-|v|+1\,..\,i+|v|-1]$, so v is a subword of s. Due to Observation 3, v is also a seed of s. Since s can be chosen arbitrarily, v is a seed of every subword of w of length $2|v|-1$.

(2) $\Rightarrow$ (1). Positions i satisfying $|v| \le i \le n-|v|+1$ are covered by full occurrences of v due to the fact that v occurs in each subword of w of length $2|v|-1$, including the one occurring as $w[i-|v|+1\,..\,i+|v|-1]$. If some position $i < |v|$ or $i > n-|v|+1$ in w is not covered by a full occurrence of v, then it must be covered by a left-overhang of v in $w[1\,..\,2|v|-1]$ or a right-overhang of v in $w[n-2|v|+2\,..\,n]$, respectively. These overhangs are also present in w. □

Corollary 2.3. *For each word w and integer k, $1 \le k \le \frac{n+1}{2}$, we have*

$$\mathrm{Seeds}_{\le k}(w) = \bigcap_{u \in \mathrm{SUB}_{2k-1}(w)} \mathrm{Seeds}_{\le k}(u).$$

Proof. Consider a length ℓ not exceeding k. Note that $\mathrm{SUB}_{2\ell-1}(w) = \bigcup_{s \in \mathrm{SUB}_{2k-1}(w)} \mathrm{SUB}_{2\ell-1}(s)$, so Lemma 2.2 yields

$$\mathrm{Seeds}_\ell(w) = \bigcap_{u \in \mathrm{SUB}_{2\ell-1}(w)} \mathrm{Seeds}_\ell(u) = \bigcap_{s \in \mathrm{SUB}_{2k-1}(w)} \bigcap_{u \in \mathrm{SUB}_{2\ell-1}(s)} \mathrm{Seeds}_\ell(u) = \bigcap_{s \in \mathrm{SUB}_{2k-1}(w)} \mathrm{Seeds}_\ell(s).$$

The equality holds for each length $\ell \le k$, which concludes the proof. □

3 REPRESENTATION THEOREM

Let v be a subword of a word w. Let us introduce a decomposition

$$w = v^- \hat{v} v^+ \tag{1}$$

such that $\hat{v}$ is the longest subword of w having v as a border. In other words,

$$\hat{v} = w[\min(Occ(v))\,..\,\max(Occ(v))+|v|-1]$$

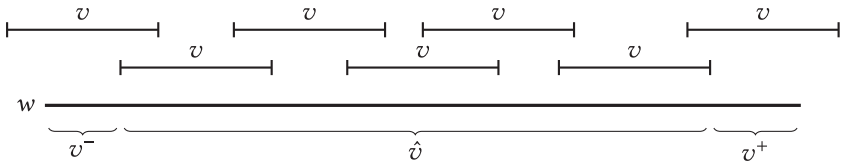

Fig. 4. Generalized occurrences of a seed v of w and the decomposition of w into v^-, $\hat{v}$, and v^+.

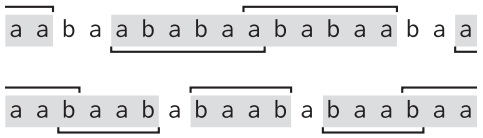

Fig. 5. The word v = ababaa is a quasiseed of w = aabaababaababaabaa because it covers $\hat{v}$ = ababaababaa, whereas the word v = baab is both a left and a right candidate of w because it is a seed of both v^-v = aabaab and vv^+ = baabaa. However, neither of them is a seed of w.

(i.e., $\hat{v}$ can be seen as the shortest fragment of w containing all full occurrences of v in w; see Figure 4).

Definition 3.1 (See Figure 5). Let us recall the decomposition $w = v^-\hat{v}v^+$. We say that v is a *quasiseed* of w if it is a cover of $\hat{v}$. If v is a seed of v^-v, then v is called a *left candidate* of w, and in case v is a seed of vv^+, then v is called a *right candidate*.

By $\mathsf{QSeeds}(w)$, $\mathsf{LCands}(w)$, and $\mathsf{RCands}(w)$, we denote the sets of quasiseeds, left candidates, and right candidates of w, respectively.

Lemma 3.2. $\mathsf{Seeds}(w) = \mathsf{LCands}(w) \cap \mathsf{QSeeds}(w) \cap \mathsf{RCands}(w)$.

Proof. We apply the characterization of Fact 4. Observe that there are natural bijections between full occurrences of v in w and in $\hat{v}$, between left-overhangs of v in w and in v^-v, and between right-overhangs of v in w and in vv^+.

Next, note that any position of w within $\hat{v}$ that is covered by an overhang of v must be located within the leading or trailing $|v|$ characters of $\hat{v}$, so it is also covered by a full occurrence of v since v is a border of $\hat{v}$. Thus, v is a quasiseed of w if and only if the positions of w within $\hat{v}$ are covered by generalized occurrences of v.

Moreover, observe that v occurs in v^-v only as a suffix, so the positions within the leading v^- of v^-v and of w can be covered by left-overhangs only. Consequently, v is a left candidate if and only if the positions within the leading v^- of w are covered by generalized occurrences of v. Symmetrically, v is a right candidate if and only if the positions within the trailing v^+ are covered by generalized occurrences of v.

Combining these three facts, we conclude that v is a seed of w if and only if it is simultaneously a quasiseed, a left candidate, and a right candidate. □

Next, we shall characterize the quasiseeds and candidates in a computationally feasible way and bound the sizes of their package representations.

3.1 Quasiseeds

For a set $X = \{x_1, \ldots, x_k\}$ of integers, $x_1 < \cdots < x_k$, let us define the value $\mathsf{maxgap}(X)$ as

$$\mathsf{maxgap}(X) = \begin{cases} 0 & \text{if } 0 \le k \le 1, \\ \max\{x_{i+1} - x_i : 1 \le i < k\} & \text{if } k \ge 2. \end{cases}$$

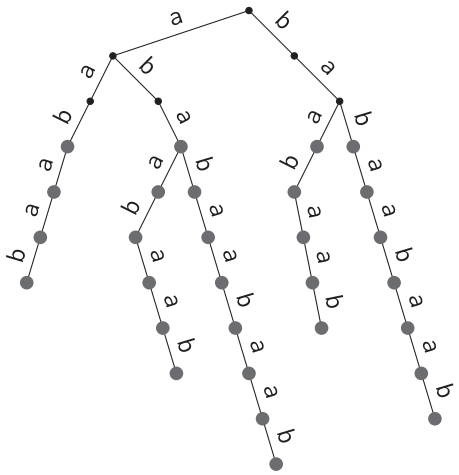

Fig. 6. Uncompressed suffix tree of w = ababaabaab with marked nodes corresponding to quasiseeds. For example, ab is not a quasiseed, since $Occ(\mathsf{ab}, w) = \{1, 3, 6, 9\}$ and $\mathsf{maxgap}(\{1, 3, 6, 9\}) = 3$; aba is a quasiseed, since $Occ(\mathsf{aba}, w) = \{1, 3, 6\}$ and $\mathsf{maxgap}(\{1, 3, 6\}) = 3$; and abab is also a quasiseed, since $Occ(\mathsf{abab}, w) = \{1\}$ and $\mathsf{maxgap}(\{1\}) = 0$ by definition.

For example, $\mathsf{maxgap}(\{1, 3, 8, 13, 17\}) = 5$. There is a known observation that relates this function to covers (e.g., see [10]). In the following, we state it directly in the context of quasiseeds.

Observation 5. *A subword v of word w is a quasiseed of w if and only if* $\mathsf{maxgap}(Occ(v, w)) \leq |v|$.

The set of quasiseeds can be easily represented in the suffix tree (see Figure 6).

Corollary 3.3. *For each equivalence class E of $\approx$, the quasiseeds contained in E form at most one package. In other words,* $\mathsf{QSeeds}(w) \cap E$ *has a package representation of size at most 1.*

Proof. Consider the shortest subword $v \in E \cap \mathsf{QSeeds}(w)$, and let $u \in E$ satisfy $|u| \geq |v|$. Due to Observation 5, $\mathsf{maxgap}(Occ(u)) = \mathsf{maxgap}(Occ(v)) \leq |v| \leq |u|$, so u is also a quasiseed. This completes the proof. □

3.2 Left Candidates

The *border table* $B[0\,..\,n]$ stores at $B[i]$ the length of the longest proper border (i.e., the longest proper subword being a border) of $w[1\,..\,i]$ (we assume $B[0] = -1$). The following fact was already shown implicitly by Christou et al. [15]; we give its proof for completeness. Let us recall the decomposition $w = v^{-}\hat{v}v^{+}$.

Lemma 3.4. *A subword v of word w is a left candidate of w if and only if $B[|v^{-}v|] \geq |v^{-}|$.*

Proof. Recall that positions within the prefix v^{-} of $v^{-}v$ can only be covered by left-overhangs of v.

Consequently, if v is a seed of $v^{-}v$, then there is a prefix of $v^{-}v$ of length at least $|v^{-}|$ equal to a suffix of v. In other words, there is a proper border of $v^{-}v$ of length at least $|v^{-}|$.

For a proof of the converse implication, suppose that $v^{-}v$ has a proper border of length at least $|v^{-}|$. Since v has only one occurrence in $v^{-}v$, the border must be a proper suffix of v. Consequently, v has a left-overhang in $v^{-}v$ of length at least $|v^{-}|$. This fragment covers all positions in v^{-}; the remaining positions in $v^{-}v$ are trivially covered by the occurrence of v as a suffix of $v^{-}v$. □

A classic property of the border table is that $0 \le B[i] \le B[i-1]+1$ holds for $1 \le i \le n$. In the following, we further characterize positions where the right inequality is strict.

Lemma 3.5. *If $B[i] \le B[i-1]$ holds for a position i of the word w, then $B[i]+B[i-1] < i-1$.*

Proof. We assume $B[i] > 0$; otherwise, the claim holds trivially: $B[i]+B[i-1] = B[i-1] < i-1$. Since $w[1\,..\,i]$ does not have a border of length $B[i-1]+1$, we must have $w[B[i-1]+1] \neq w[i] = w[B[i]]$, so $B[i-1]+1-B[i]$ is not a period of $w[1\,..\,i-1]$ despite the fact that both $i-B[i]$ and $i-1-B[i-1]$ are periods of this prefix. By the periodicity lemma, this yields $(i-B[i])+(i-1-B[i-1])-1 > i-1$, so $B[i]+B[i-1] < i-1$ holds as claimed. □

Let v be a subword of w, and let $v = v'a$ where $a \in \Sigma$. We say that v is a *critical left candidate of* w if it is a left candidate of w, but

(1) v' is not a left candidate of w (in particular, v' can be the empty word) or
(2) $\min(Occ(v')) < \min(Occ(v))$.

Let

$$\mathsf{Active}(w) \;=\; \left\{ j \in [1\,..\,n] \;:\; B[j-1]+1 = B[j] \le \tfrac{j-1}{2} \right\}.$$

The following lemma is the main technical contribution in the characterization of left candidates.

Lemma 3.6. *A fragment $w[i\,..\,j]$ is*

(a) the leftmost occurrence of a left candidate if and only if $i-1 \le B[j] \le j-i$;
(b) the leftmost occurrence of a critical left candidate if and only if $i = B[j]+1$ and $j \in \mathsf{Active}(w)$.

Proof. (a) ($\Leftarrow$). Let us recall the decomposition $w = v^-\hat{v}v^+$. If $w[i\,..\,j]$ is the leftmost occurrence of a left candidate v, then $w[1\,..\,j] = v^-v$ and $B[j] \ge |v^-| = i-1$ by Lemma 3.4. Moreover, $B[j] < |v| = j-i+1$, because $w[i-j+B[j]\,..\,B[j]]$ would otherwise be an earlier occurrence of v. These two conditions yield $i-1 \le B[j] \le j-i$.

($\Rightarrow$). Conversely, assume that

$$i-1 \le B[j] \le j-i. \tag{2}$$

Suppose for a proof by contradiction that $v = w[i\,..\,j]$ has an earlier occurrence at position i'—that is, $w[i'\,..\,i'+j-i] = w[i\,..\,j]$ for some position $i' < i$. Let $w' = w[i'\,..\,j]$. The word v is a border of w', so w' has a period $i-i'$, which we denote by p_1. The shortest period of the whole prefix $w[1\,..\,j]$, hence a period of w', is $j-B[j]$, which we denote by p_2. By the first inequality of (2),

$$p_1+p_2 \;=\; i-i'+j-B[j] \;\le\; j-i'+1 \;=\; |w'|.$$

Hence, p_1 and p_2 satisfy the assumption of the periodicity lemma and w' has period $\gcd(p_1,p_2)$, which we denote by p. Moreover, by the second inequality of (2),

$$p_1 \;=\; i-i' < i \;\le\; j-B[j] \;=\; p_2,$$

so $p < p_2$. Hence, $w[1\,..\,j]$ has period p_2, whereas its suffix $w[i'\,..\,j]$ has period p dividing p_2. Consequently, $w[1\,..\,j]$ has period p,[2] which contradicts the choice of p_2 as the shortest period of $w[1\,..\,j]$.

Therefore, $w[i\,..\,j]$ indeed is the leftmost occurrence of v and $v^-v = w[1\,..\,j]$. Due to $B[j] \ge i-1 = |v^-|$, we conclude that v is a left candidate by Lemma 3.4.

[2] One can show by induction that $w[k] = w[k+p]$ for $1 \le k \le j-p$. In the base case of $k \ge i'$, this follows from p being a period of $w[i'\,..\,j]$. For $k < i'$, on the other hand, we have $w[k] = w[k+p_2] = w[k+p\cdot\frac{p_2}{p}] = \cdots = w[k+p]$ by the inductive hypothesis (applied to $k+\ell p$ for $1 \le \ell < \frac{p_2}{p}$) and since p_2 is a period of $w[1\,..\,j]$ and a multiple of p.

(b) ($\Rightarrow$). Let us assume that $B[j-1]+1 = B[j] \leq \frac{j-1}{2}$ and $i = B[j]+1$.
Note that

$$i - 1 = B[j] = 2B[j] - B[j] \leq j - 1 - B[j] = j - i.$$

By part (a), $w[i\,..\,j]$ is therefore the leftmost occurrence of a left candidate. However, $B[j-1]+1 = B[j] = i-1$, so $w[i\,..\,j-1]$ is not the leftmost occurrence of a left candidate. Thus, $w[i\,..\,j]$ is the leftmost occurrence of a critical left candidate.

($\Leftarrow$). Let us assume that $w[i\,..\,j]$ is the leftmost occurrence of a critical left candidate. Part (a) yields $i - 1 \leq B[j] \leq j - i$ (since $w[i\,..\,j]$ is the leftmost occurrence of a left candidate) and $B[j-1] < i-1$ or $B[j-1] \geq j-i$ (since $w[i\,..\,j-1]$ is not).

In the latter case, we have $B[j-1] + B[j] \geq j - i + i - 1 = j - 1$, so $B[j] = B[j-1]+1$ holds by Lemma 3.5. This yields a contradiction:

$$j - i \geq B[j] = B[j-1] + 1 \geq j - i + 1.$$

Thus, the only possibility is that $B[j-1] < i-1$, i.e., $B[j-1] \leq i-2$. We then have

$$B[j] \leq B[j-1] + 1 \leq i - 1 \leq B[j],$$

so $B[j-1]+1 = B[j] = i-1$. Furthermore,

$$B[j] = \tfrac{1}{2}(B[j] + B[j]) \leq \tfrac{1}{2}(j - i + i - 1) = \tfrac{j-1}{2}$$

holds as claimed. □

For a table $A[0\,..\,n+1]$, assuming $A[n+1] = -\infty$, define the *nearest smaller value* table $\mathcal{N}_A$ such that for $0 \leq i \leq n$ we have $\mathcal{N}_A[i] = \min\{j > i : A[j] < A[i]\}$.

Lemma 3.7. *Packages* $\mathsf{pack}(B[j]+1, j, \mathcal{N}_B[j]-1)$ *for* $j \in \mathsf{Active}(w)$ *form a package representation of the family of left candidates of* w*. This representation (of size at most* n*) can be computed in* $O(n)$ *time.*

Proof. First, we shall prove that for each $j \in \mathsf{Active}(w)$, the fragments $w[B[j]+1\,..\,k]$ for $j \leq k \leq \mathcal{N}_B[j]-1$ are leftmost occurrences of left candidates. To apply the characterization of Lemma 3.6(a), we need to show that $B[j] \leq B[k] \leq k - B[j] - 1$. The first inequality follows directly from the definition of the $\mathcal{N}_B$ table, whereas for the second one, we observe that $B[j] + B[k] \leq 2B[j] + k - j \leq j - 1 + k - j = k - 1$ holds due to $B[j] \leq \frac{j-1}{2}$ and the classic property of the border table.

Consequently, the packages are disjoint and consist of left candidates only.

It remains to prove that we do not leave any left candidate behind. Suppose that $w[i\,..\,k]$ is the leftmost occurrence of a left candidate. Repeatedly trimming the trailing character, we can reach a critical left candidate whose leftmost occurrence is $w[i\,..\,j]$ (for $j \leq k$). By Lemma 3.6(b), we have that $j \in \mathsf{Active}(w)$ and $i = B[j]+1$. However, since $w[i\,..\,k']$ is the leftmost occurrence of a left candidate for all $j \leq k' \leq k$, Lemma 3.6(a) yields $B[k'] \geq i - 1 = B[j]$ for $j \leq k' \leq k$. Consequently, $\mathcal{N}_B[j] > k$. Thus, $w[i\,..\,k] \in \mathsf{pack}(B[j]+1, j, \mathcal{N}_B[j]-1)$ indeed belongs to one of the packages we created.

The size of the package representation is $|\mathsf{Active}(w)| \leq n$. As for the $O(n)$-time construction algorithm, we first build the border table B [17, 18, 48] and its nearest smaller value table $\mathcal{N}_B$ (using a folklore algorithm; e.g., see [11]). Then, for each position j, we test in constant time whether $j \in \mathsf{Active}(w)$, and, if so, we retrieve the corresponding package. □

3.3 Right Candidates

For word w, let us define a *reverse border array* $B^R[1\,..\,n]$ such that $B^R[i]$ is the length of the longest proper border of $w[i\,..\,n]$.

Lemma 3.8. *A subword v of a word w is a right candidate of w if and only if $B^R[n - |vv^+| + 1] \geq |v^+|$.*

Proof. Follows from Lemma 3.4 by the symmetry between B and B^R, as well as left and right candidates. □

Even though Lemma 3.8 for right candidates and Lemma 3.4 for left candidates are symmetric, the former allows us to represent right candidates on each edge of the suffix tree of w as a single package. Thus, a representation for right candidates is much simpler to compute than the representation for left candidates.

Lemma 3.9. *The intersection of* $\mathsf{RCands}(w)$ *with a single equivalence class of $\approx$ forms at most one package. Moreover, a package representation (of size at most $2n$) of the set* $\mathsf{RCands}(w)$ *can be computed in $O(n)$ time.*

Proof. Let us consider an equivalence class E of the relation $\approx$, with $P = Occ(v)$ for $v \in E$, and denote

$$\ell(E) = n - \max(P) + 1 - B^R[\max(P)].$$

We will show that $v \in E$ is a right candidate if and only if $|v| \geq \ell(E)$. By Lemma 3.8, v is a right candidate if and only if $B^R[n - |vv^+| + 1] \geq |v^+|$. However, $|vv^+| = n - \max(P) + 1$ and $|v^+| = n - \max(P) + 1 - |v|$, so the two conditions are equivalent.

For every equivalence class E that corresponds to an edge of the suffix tree of w, we will show that $\mathsf{RCands}(w) \cap E$ is either empty or is a single package. We will also show how these intersections can all be computed in $O(n)$ total time.

We construct the array B^R [17, 18, 48] and the suffix tree of w. Then we traverse the suffix tree bottom up and compute $\max(Occ(v))$ for every subword v that corresponds to an explicit node as the maximum of the values of its explicit children. This lets us compute $\ell(E)$ for the equivalence class $E = \mathsf{pack}(i, j_1, j_2)$ containing v. We have proved that the right candidates in E are precisely words in E of length $\ell(E)$ or more. If $\ell(E) > |w[i\,..\,j_2]|$, there are no right candidates in E. Otherwise, the right candidates form a package

$$\mathsf{pack}(i, \max(j_1, i + \ell(E) - 1), j_2).$$ □

3.4 Representation Theorem for Seeds

Theorem 6. *For a word w of length n, the set* $\mathsf{Seeds}(w)$ *has a package representation of size at most $3n$.*

Proof. By Lemma 3.2, we have $\mathsf{Seeds}(w) = \mathsf{LCands}(w) \cap \mathsf{QSeeds}(w) \cap \mathsf{RCands}(w)$.

First, we shall prove that the package representation of $\mathsf{QSeeds}(w) \cap \mathsf{RCands}(w)$ consists of at most $2n$ packages. Indeed, for each equivalence class E of $\approx$, both $\mathsf{QSeeds}(w) \cap E$ and $\mathsf{RCands}(w) \cap E$ form at most one package. The intersection of two packages forms at most one package, so $\mathsf{QSeeds}(w) \cap \mathsf{RCands}(w)$ has a package representation with at most one package per equivalence class (i.e., of total size at most $2n$).

By Lemma 3.7, $\mathsf{LCands}(w)$ has a package representation of size at most n.

Finally, Lemma 2.1 implies that $\mathsf{Seeds}(w)$ has a package representation of size at most $3n$. □

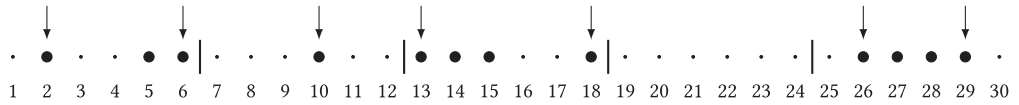

Fig. 7. For $n = 30$, $A = \{2, 5, 6, 10, 13, 14, 15, 18, 26, 27, 28, 29\}$, and $B = \{2, 6, 10, 13, 18, 26, 29\}$, we have $\mathsf{refine}_6(A) = B$. From each block of length $\ell = 6$ at most two elements are retained—the extreme ones.

4 COMPUTING LONG SEEDS

In this section, we provide a linear-time algorithm computing seeds of length $\Theta(n)$. Formally, we consider the following operation for $\ell = Cn$, for some constant C:

LONG-SEEDS(ℓ, w): Computes a package representation of $\mathsf{Seeds}_{\geq \ell}(w)$.

Let us denote by $\mathsf{QSeeds}_{\geq \ell}(w)$ the set of all quasiseeds of w with lengths at least ℓ. Our implementation is based on Theorem 6, and it uses the suffix tree of w to determine $\mathsf{QSeeds}_{\geq \ell}(w)$.

Let us partition the positions of w into a family $\mathcal{F}$ of $O(n/\ell)$ disjoint *blocks* of length at most ℓ each. For a set of positions X, we define its refined version:

$$\mathsf{refine}_\ell(X) \;=\; \bigcup_{Y \in \mathcal{F} \,:\, Y \cap X \neq \emptyset} \{\min(Y \cap X),\, \max(Y \cap X)\}$$

(see Figure 7). Note that $|\mathsf{refine}_\ell(X)| \leq 2|\mathcal{F}| = O(n/\ell) = O(1)$.

We can now make the following observation.

Observation 7. *A subword v of w such that $|v| \geq \ell$ is a quasiseed if and only if* $\mathsf{maxgap}(\mathsf{refine}_\ell(Occ(v))) \leq |v|$.

We traverse the suffix tree of w bottom up, computing $\mathsf{refine}_\ell(Occ(v))$ in constant time for each subword v whose locus is an explicit node (i.e., a node that is stored explicitly in the suffix tree).

Fact 8. *The sets* $\mathsf{refine}_\ell(Occ(v))$ *for all explicit nodes v of the suffix tree can be computed in $O(n)$ time.*

Proof. We compute $\mathsf{refine}_\ell(Occ(v))$ for each explicit node v and store it as a sorted list. For this, we start with an empty set and consider all explicit children u of v. For each child, we merge the current list with $\mathsf{refine}_\ell(Occ(u))$, removing elements that are not extremes in their blocks. This takes $O(|\mathcal{F}|) = O(1)$ time for each edge of the suffix tree, which gives $O(n)$ time in total. □

Lemma 4.1 (LONG-SEEDS Implementation). *For a threshold $\ell = \Theta(n)$ and a word w of length n, LONG-SEEDS(ℓ, w) can be implemented in $O(n)$ time.*

Proof. First, we compute a package representation of the family $\mathsf{QSeeds}_{\geq \ell}(w)$ of long quasiseeds. Consider an equivalence class $E = \mathsf{pack}(i, j_1, j_2)$ of the relation $\approx$. Let P be the common occurrence set $Occ(v)$ for $v \in E$. The longest subword in each package is represented by an explicit node, so the procedure of Fact 8 computes $\mathsf{refine}_\ell(P)$. Let us define $\ell' := \max(\ell, \mathsf{maxgap}(\mathsf{refine}_\ell(P)))$. By Observation 7, a subword $v \in E$ is a long quasiseed if and only if $|v| \geq \ell'$. If $j_2 - i + 1 < \ell'$, there are no such quasiseeds. Otherwise, we report a package

$$E \cap \mathsf{QSeeds}_{\geq \ell}(w) = \mathsf{pack}(i, \max(j_1, i + \ell' - 1), j_2).$$

Finally, we apply Lemma 3.2 and compute $\mathsf{Seeds}_{\geq \ell}(w) = \mathsf{QSeeds}_{\geq \ell}(w) \cap \mathsf{LCands}(w) \cap \mathsf{RCands}(w)$ using Lemma 2.1 to implement the intersection. Linear-size package representations of left candidates and right candidates are constructed using Lemmas 3.7 and 3.9, respectively. The total running time is $O(n)$. □

Table 1. Subword complexity of w and $\beta_k(w)$ for w = abababaaba

k	1	2	3	4	5	6	7	8
$w[k]$	a	b	a	b	a	a	b	a
$\lvert\mathrm{SUB}_k(w)\rvert$	2	3	4	5	4	3	2	1
$\beta_k(w)$	2	4	6	8	8	8	8	8

We also use the following operation that can be computed using LONG-SEEDS. Let us recall the notation Seeds_I from Section 2.2:

MID-SEEDS(I, w): Computes a package representation of $\mathrm{Seeds}_I(w)$.

Our implementation of MID-SEEDS(I, w) uses a reduction to Lemmas 4.1 and 2.1, and its running time is linear provided that $I = [\ell\,..\,r]$ is *balanced* (i.e., if the ratio $\frac{r}{\ell}$ in bounded by a constant). In our algorithm, this operation is used only for balanced intervals with $r \le 8\ell$. If the interval $I = [\ell\,..\,r]$ were not balanced, this function could be implemented in $O(\frac{r}{\ell}n)$ time.

Lemma 4.2 (MID-SEEDS Implementation). *For an interval $I = [\ell\,..\,r]$ and a word w of length n, MID-SEEDS(I, w) can be implemented in $O(n)$ time if $r = O(\ell)$.*

Proof. We construct a family R of fragments of length $4r$ covering w with overlaps of size $2(r-1)$ (the last fragment might be shorter). Note that the total length of these fragments is at most $2n$. Furthermore, observe that $\mathrm{SUB}_{2r-1}(w) = \bigcup_{s\in R}\mathrm{SUB}_{2r-1}(s)$, so Corollary 2.3 yields

$$\mathrm{Seeds}_{\le r}(w) = \bigcap_{u\in\mathrm{SUB}_{2r-1}(w)}\mathrm{Seeds}_{\le r}(u) = \bigcap_{s\in R}\bigcap_{u\in\mathrm{SUB}_{2r-1}(s)}\mathrm{Seeds}_{\le r}(u) = \bigcap_{s\in R}\mathrm{Seeds}_{\le r}(s).$$

In particular, $\mathrm{Seeds}_I(w) = \bigcap_{s\in R}\mathrm{Seeds}_I(s)$. For each $s \in R$, we apply Lemma 4.1 to determine a package representation of $\mathrm{Seeds}_{\ge\ell}(s)$, and we filter out seeds of length greater than r. This takes $O(|s|)$ time for each $s \in R$, which is $O(n)$ in total. Finally, we combine the package representations in $O(n)$ time using Lemma 2.1. □

5 RELATION AMONG SEEDS, COMPRESSION, AND SUBWORD COMPLEXITY

Recall that subword complexity of a word w maps each length k to the number $|\mathrm{SUB}_k(w)|$ of length-k subwords of w. Since $|\mathrm{SUB}_k(w)|$ is not monotone in general, we define a non-decreasing sequence $(\beta_k)_{k=1}^n$ with

$$\beta_k(w) = |\mathrm{SUB}_k(w)| + k - 1$$

(see Table 1).

Lemma 5.1. *The sequence $|\mathrm{SUB}_m(w)|$ (consequently, also the sequence $\beta_m(w)$) can be computed in linear time.*

Proof. Each equivalence class $E = \mathrm{pack}(i, j_1, j_2)$ contributes a single subword in $\mathrm{SUB}_m(w)$ for each $m \in [j_1 - i + 1\,..\,j_2 - i + 1]$. We obtain at most $2n$ such intervals; $|\mathrm{SUB}_m(w)|$ is then the number of intervals that contain the element m. This quantity can be computed in $O(n)$ time using an auxiliary array A of size n (initially set to zeroes). For an interval $[a\,..\,b]$, $A[a]$ is incremented and $A[b+1]$ is decremented. Then $|\mathrm{SUB}_m(w)| = A[1] + \cdots + A[m]$. □

A connection between the subword complexity and the existence of seeds of certain lengths is crucial in this work. More precisely, each seed provides an upper bound on the values $\beta_k(w)$, as shown in the following Lemma 5.2. This lemma yields a gap in the feasible lengths of seeds of w.

Seeds of length $\frac{1}{6}n$ or more can be determined using the LONG-SEEDS procedure, so it allows us to focus on computing relatively short seeds.

Lemma 5.2 [Gap Lemma]. *If $\beta_k(w) > \frac{2}{3}n$ for some $1 \le k \le \frac{1}{6}n$, then w has no seed v whose length satisfies $2k-2 \le |v| \le \frac{1}{6}n$.*

Proof. We shall prove that if v is a seed of w, then $\beta_k(w) \le |v| + (k-1)\frac{n+|v|-2(k-1)}{|v|}$ for $k \le |v|$.

Consider length-k fragments starting at positions $i, \ldots, i+\ell$, where $\ell < |v|$. We claim that at most $k-1$ of these fragments are not covered by single occurrences of v. Let $w[i' \,..\, i'+k-1]$, for $i \le i' \le i+\ell$, be the first such fragment that is not covered by any single occurrence of v. If i' does not exist or $i'+k-1 > i+\ell$, we are done. Otherwise, the occurrence of v covering position $i'+k-1$ must start at some position j, $i' < j < i'+k$. Consequently, the length-k fragments starting at positions i'', $j \le i'' \le j+|v|-k$ are all covered by this occurrence of v. We are left with at most $k-1$ remaining length-k fragments (starting at positions i'' such that $i' \le i'' < j$ or $j+|v|-k < i'' \le i+\ell$).

Thus, out of $n-k+1$ length-k fragments of w, the number of fragments not covered by single occurrences of v is at most

$$\begin{aligned}(k-1)\left\lfloor \tfrac{n-k+1}{|v|} \right\rfloor + \min((n-k+1) \bmod |v|, k-1) &\le (k-1)\tfrac{n-k+1}{|v|} + \tfrac{|v|-(k-1)}{|v|}(k-1)\\ &= (k-1)\tfrac{n+|v|-2(k-1)}{|v|}.\end{aligned}$$

As a result, $|\mathrm{SUB}_k(w) \setminus \mathrm{SUB}_k(v)| \le (k-1)\frac{n+|v|-2(k-1)}{|v|}$, and we obtain the claimed upper bound on $\beta_k(w)$:

$$\begin{aligned}\beta_k(w) = k-1+|\mathrm{SUB}_k(w)| &= k-1+|\mathrm{SUB}_k(v)| + |\mathrm{SUB}_k(w) \setminus \mathrm{SUB}_k(v)|\\ &\le |v| + (k-1)\tfrac{n+|v|-2(k-1)}{|v|}.\end{aligned}$$

If $2k-2 \le |v| \le \frac{1}{6}n$, then, by monotonicity of $\beta_k(w)$ with respect to k, we conclude that

$$\beta_k(w) \le \beta_{\frac{1}{2}|v|+1}(w) \le |v| + \tfrac{|v|}{2}\tfrac{n+|v|-|v|}{|v|} = |v| + \tfrac{1}{2}n \le \tfrac{1}{6}n + \tfrac{1}{2}n = \tfrac{2}{3}n,$$

which contradicts the hypothesis of the lemma. □

Let us thus focus on computing seeds of length at most $2k-3$. Due to the characterization of Lemma 2.2, we may ignore some regions of w, as long as we do not miss any subwords of certain lengths (so that some occurrences of a seed can be missed but not all of them). To formalize this intuition, we define an operation $\mathrm{COMPR}_k(w)$, which yields a set S of subwords of w such that $\mathrm{SUB}_k(w) = \bigcup_{s \in S} \mathrm{SUB}_k(s)$.

Let us take the set of fragments Y that are leftmost occurrences of subwords in $\mathrm{SUB}_k(w)$. While there are two fragments in Y covering the same position or containing one of any two consecutive positions of w each, we remove them from Y and include in Y the minimal fragment containing both of them. (Formally, while Y contains fragments $w[i \,..\, j]$ and $w[i' \,..\, j']$ such that $[i-1 \,..\, j] \cap [i'-1 \,..\, j'] \neq \emptyset$, we remove both from Y and include a fragment $w[\min(i,i') \,..\, \max(j,j')]$.) Finally, the subwords indicated by the resulting set of fragments give the family $\mathrm{COMPR}_k(w)$ (see Figure 8 for an example).

Recall that our motivation is to reduce computing short seeds in w to the analogous operation on each $s \in \mathrm{COMPR}_k(w)$. This is illustrated by the following result.

Lemma 5.3 [Reduction Lemma]. *Consider a word w of length n. For every integer k, $1 \le k \le \frac{n+1}{2}$, we have*

$$\mathrm{Seeds}_{\le k}(w) = \bigcap_{s \in \mathrm{COMPR}_{2k-1}(w)} \mathrm{Seeds}_{\le k}(s).$$

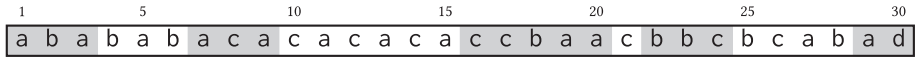

Fig. 8. For a word $w = \mathsf{abababacacacacaccbaacbbcbcabad}$, we have $\mathrm{COMPR}_2(w) = \{w[1\,..\,3], w[7\,..\,9], w[16\,..\,20], w[22\,..\,24], w[29\,..\,30]\} = \{\mathsf{aba}, \mathsf{aca}, \mathsf{ccbaa}, \mathsf{bbc}, \mathsf{ad}\}$. And indeed $\mathrm{SUB}_2(\mathsf{aba}) \cup \mathrm{SUB}_2(\mathsf{aca}) \cup \mathrm{SUB}_2(\mathsf{ccbaa}) \cup \mathrm{SUB}_2(\mathsf{bbc}) \cup \mathrm{SUB}_2(\mathsf{ad}) = \{\mathsf{ab}, \mathsf{ba}\} \cup \{\mathsf{ac}, \mathsf{ca}\} \cup \{\mathsf{cc}, \mathsf{cb}, \mathsf{ba}, \mathsf{aa}\} \cup \{\mathsf{bb}, \mathsf{bc}\} \cup \{\mathsf{ad}\} = \mathrm{SUB}_2(w)$.

Proof. Note that $\mathrm{SUB}_{2k-1}(w) = \bigcup_{s\in\mathrm{COMPR}_{2k-1}(w)} \mathrm{SUB}_{2k-1}(s)$. Hence, Corollary 2.3 yields

$$\mathrm{Seeds}_{\le k}(w) = \bigcap_{u\in\mathrm{SUB}_{2k-1}(w)} \mathrm{Seeds}_{\le k}(u) = \bigcap_{s\in\mathrm{COMPR}_{2k-1}(w)} \bigcap_{u\in\mathrm{SUB}_{2k-1}(s)} \mathrm{Seeds}_{\le k}(u) = \bigcap_{s\in\mathrm{COMPR}_{2k-1}(w)} \mathrm{Seeds}_{\le k}(s).$$

This concludes the proof. □

The values $\beta_k(w)$ can be used to bound the total length of words $s \in \mathrm{COMPR}_k(w)$, denoted $\|\mathrm{COMPR}_k(w)\|$.

Lemma 5.4 [Compression Lemma]. *For each word w and integer k, $1 \le k \le \frac{n+1}{2}$, we have*

$$\|\mathrm{COMPR}_k(w)\| \le \beta_{2k-1}(w).$$

Proof. Let P be the set of positions covered by the leftmost occurrences of subwords from $\mathrm{COMPR}_k(w)$. By construction, $\|\mathrm{COMPR}_k(w)\| = |P|$ and $i \in P$ if and only if i is included in the leftmost occurrence of some subword $s \in \mathrm{SUB}_k(w)$. If $k \le i \le n-k+1$, then i is the midpoint of a length-$(2k-1)$ fragment $w[i-k+1\,..\,i+k-1]$, which covers all length-k fragments of w containing position i, including the leftmost occurrence of $s \in \mathrm{SUB}_k(w)$. Thus, $w[i-k+1\,..\,i+k-1]$ is also the leftmost occurrence of the corresponding subword of length $2k-1$. This yields an injective mapping from the set of positions $i \in P \cap [k\,..\,n-k+1]$ to the family $\mathrm{SUB}_{2k-1}(w)$. The remaining positions ($i < k$ and $i > n-k+1$) account for the extra term $2k-2 = \beta_{2k-1}(w) - |\mathrm{SUB}_{2k-1}(w)|$ in the upper bound. □

From the last two lemmas, we obtain the following corollary that conveys the intuition behind the main structural theorem in the following section.

Corollary 5.5. *Let $1 \le k \le \frac{n+3}{4}$ and $S = \mathrm{COMPR}_{2k-1}(w)$. Then*

$$\mathrm{Seeds}_{\le k}(w) = \bigcap_{s\in S} \mathrm{Seeds}_{\le k}(s) \quad \text{and} \quad \|S\| \le \beta_{4k-3}(w).$$

6 MAIN ALGORITHM

The main algorithm is a recursive procedure based on a structural theorem that combines the results of the previous section.

Theorem 9 [Decomposition Theorem]. *Consider a word w of length n. If $n \le 6$ or $|Alph(w)| > \frac{2}{3}n$, then $\mathrm{Seeds}(w) = \mathrm{Seeds}_{\ge \frac{1}{6}n}(w)$. Otherwise, we can compute in linear time a balanced interval I and a family S of subwords of w such that*

$$\textbf{(A)}\ \mathrm{Seeds}(w) = \mathrm{Seeds}_{\ge \frac{1}{6}n}(w) \cup \mathrm{Seeds}_I(w) \cup \bigcap_{s\in S} \mathrm{Seeds}_{\le k}(s) \qquad \textit{and} \qquad \textbf{(B)}\ \|S\| \le \tfrac{2}{3}n.$$

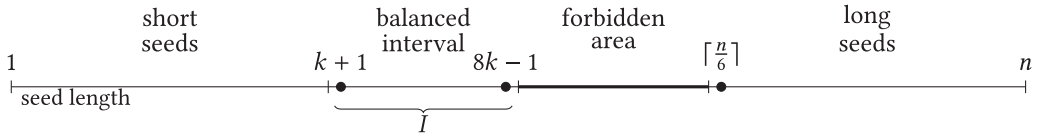

Fig. 9. Illustration of Theorem 9 in case $8k \le \lceil \frac{n}{6} \rceil$. There is no seed whose length would be in the forbidden area. The computation of all seeds is split into recursive computation of short seeds (of length at most k) in subwords $s \in S$, of seeds with lengths in a balanced interval I, and of long seeds.

1					5					10					15					20					25					30
a	b	a	b	a	b	a	c	a	c	a	a	c	a	c	a	c	c	b	a	a	c	b	b	c	b	c	a	b	a	d
1	1	0	0	0	0	1	1	0	0	0	0	0	0	0	1	1	0	1	0	0	1	1	0	0	0	0	0	1	0	
1	1	1	0	0	0	1	1	1	0	0	0	0	0	0	1	1	1	1	1	0	1	1	1	0	0	0	0	1	1	

Fig. 10. Computation of $\mathrm{COMPR}_2(w)$ for w as in Figure 8. The second line contains the array A after the leftmost occurrences of length-2 factors are marked, and the third line shows the final array A that corresponds to the letters shown in gray.

Proof. We take

$$k = \max\left\{\ell \,:\, 1 \le \ell < \lceil \tfrac{n}{6} \rceil \text{ and } \beta_{4\ell-3}(w) \le \tfrac{2}{3}n\right\},$$
$$S = \mathrm{COMPR}_{2k-1}(w),$$
$$I = [k+1\,..\,\min(8k, \lceil \tfrac{n}{6} \rceil) - 1].$$

As for the first part of the statement, note that $\mathrm{Seeds}(w) = \mathrm{Seeds}_{\ge \frac{1}{6}n}(w)$ holds trivially if $n \le 6$. On the other hand, if $|Alph(w)| = \beta_1(w) > \frac{2}{3}n$, then the equality follows directly from the gap lemma (Lemma 5.2).

Hence, we can assume further that $n > 6$ and $\beta_1(w) \le \frac{2}{3}n$.

Correctness of (B). Now we can guarantee that k is well defined and that it satisfies $\beta_{4k-3}(w) \le \frac{2}{3}n$. Then the compression lemma (Lemma 5.4) implies that S satisfies condition **(B)**.

Correctness of (A). To prove condition **(A)**, we observe that the reduction lemma (Lemma 5.3) yields $\mathrm{Seeds}_{\le k}(w) = \bigcap_{s \in S} \mathrm{Seeds}_{\le k}(s)$. Thus, it is enough to prove that

$$\mathrm{Seeds}_{\ge k+1}(w) = \mathrm{Seeds}_{\ge \frac{1}{6}n}(w) \cup \mathrm{Seeds}_I(w)$$

(i.e., that there is no seed v with $8k \le |v| < \lceil \frac{n}{6} \rceil$). This claim holds trivially for $k = \lceil \frac{n}{6} \rceil - 1$. Otherwise, $\beta_{4k+1} > \frac{2}{3}n$, and the claim follows directly from the gap lemma (Lemma 5.2).

Computing k and S. By Lemma 5.1, the sequence $\beta_m(w)$ can be computed in linear time. This allows us to compute k by definition in $O(n)$ time.

Let us show how we compute $S = \mathrm{COMPR}_p(w)$ for $p = 2k-1$. We will iterate over all distinct subwords in $\mathrm{SUB}_k(w)$ and mark the starting positions of their leftmost occurrences. An auxiliary binary array $A[1\,..\,n]$ is used, initially set to 0s. We iterate over the equivalence classes $E = \mathrm{pack}(i, j_1, j_2)$ of $\approx$—that is, over the edges of the suffix tree of w—in a bottom-up order. For every such class, we compute the common value $x = \min(Occ(v))$ for $v \in E$ (as in the proof of Lemma 3.9). If an equivalence class E contains a subword of length p (i.e., if $p \in [j_1 - i + 1\,..\,j_2 - i + 1]$), we set $A[x]$ to 1.

Next, for every position x such that $A[x] = 1$, we set $A[x']$ to 1 for every $x' \in [x+1\,..\,\min(y, x+p-1)]$, where $y > x$ is the next 1-position in A or n if no such position exists. See Figure 10 for an example.

ALGORITHM 1: Recursive procedure SEEDS(w)

Input: A word w of length n.
Output: An $O(n)$-size package representation of Seeds(w).

if $n \le 6$ **or** $|Alph(w)| > \frac{2}{3}n$ **then return** LONG-SEEDS($\lceil \frac{n}{6} \rceil, w$)

Let I, S be as in the proof of the decomposition theorem

foreach $v \in S$ **do**
    $R_v :=$ SEEDS(v)

$R :=$ Combine($\{ R_v : v \in S \}$)

Remove from R seeds of length at least min(I)

return $R \cup$ LONG-SEEDS($\lceil \frac{n}{6} \rceil, w$) $\cup$ MID-SEEDS(I, w)
        (The three package representations contain distinct seeds)

After these two phases, we build a subword $w[x \,..\, y] \in S$ out of each maximal interval of 1-positions $[x \,..\, y]$ in A. □

The algorithm computing seeds relies on Theorem 9, with the LONG-SEEDS procedure applied to compute $\mathrm{Seeds}_{\ge \frac{1}{6}n}(n)$ and recursive calls made to determine Seeds(s) for each $s \in S$. Finally, a package representation of $\mathrm{Seeds}_I(w)$ is computed using MID-SEEDS.

Theorem 10 [Main Result]. *An $O(n)$-size package representation of the set* Seeds(w) *of all seeds of a given length-n word w can be found in $O(n)$ time. In particular, a shortest seed and the total number of seeds can be computed within the same time complexity.*

Proof. Correctness of the algorithm SEEDS follows immediately from Theorem 9. To bound the running time, let us denote by $T(n)$ the maximum number of operations performed by the SEEDS function executed for a word of length n. Due to Theorem 9 and Lemmas 2.1, 4.2, and 4.1,

$$T(n) = O(n) + \sum_i T(n_i), \quad \text{where } \sum_i n_i \le \tfrac{2}{3}n.$$

This recurrence yields $T(n) = O(n)$. □

7 OFFLINE WEIGHTED ANCESTOR QUERIES

In the weighted ancestor problem, introduced by Farach and Muthukrishnan [20] (also see [25]), we consider a rooted tree T with an integer weight function weight defined on the nodes. The weight of the root must be zero, and the weight of any other node must be strictly larger than the weight of its parent. A classic example is any compacted trie with the weight of a node defined as the length of its value.

The weighed ancestor queries, given a node v and an integer value $\ell \le \mathrm{weight}(v)$, ask for the highest ancestor μ of v such that $\mathrm{weight}(\mu) \ge \ell$—that is, such an ancestor μ that $\mathrm{weight}(\mu) \ge \ell$ and $\mathrm{weight}(\mu)$ is the smallest possible. We denote the node μ as $\mathtt{ancestor}(v, \ell)$.

Weighted ancestor queries in the online setting can be answered in $O(\log \log n)$ time after $O(n)$-time preprocessing [3]. In the special case of the weighted tree being a suffix tree of a word, they can be answered in $O(1)$ time with a data structure of $O(n)$ space [25]. Nevertheless, no efficient construction of this data structure is known. In the following, we show that if all queries are given offline and the weights are polynomially bounded, then q queries can be answered in $O(n + q)$ time—that is, in time that is linear in the input size.

Our subroutine has already found other applications (e.g., see [1, 2, 9]).

Let us first recall the classic union-find data structure. It maintains a partition $\mathcal{S}$ of $[1\,..\,n]$. Each set $S \in \mathcal{S}$ has an identifier $\mathrm{id}(S) \in S$. Initially, $\mathcal{S}$ is a partition into singletons and $\mathrm{id}(\{i\}) = i$ for $1 \le i \le n$.

The union-find data structure supports $\texttt{find}(i)$ queries that, given an integer $i \in [1\,..\,n]$, return the identifier $\mathrm{id}(S)$ of the set $S \in \mathcal{S}$ containing i. Moreover, a $\texttt{union}(i_1, i_2)$ operation, given integers $i_1, i_2 \in [1\,..\,n]$, replaces the sets $S_1, S_2 \in \mathcal{S}$ such that $i_1 \in S_1$ and $i_2 \in S_2$ with their union $S_1 \cup S_2$. Note that $\texttt{union}(i_1, i_2)$ is void if $S_1 = S_2$ (i.e., if $\texttt{find}(i_1) = \texttt{find}(i_2)$).

We will only encounter *linear* union-find instances, in which sets $S \in \mathcal{S}$ are formed by consecutive integers and $\mathrm{id}(S) = \min(S)$. In other words, $\texttt{union}(i_1, i_2)$ is allowed for $i_1 < i_2$ only if $\texttt{find}(i_1) = \texttt{find}(\texttt{find}(i_2) - 1)$. For such instances, the union-find operations can be implemented in amortized $O(1)$ time.

Lemma 7.1 [Gabow and Tarjan [24]]. *A sequence of m linear union-find operations on a partition of $[1\,..\,n]$ can be implemented in $O(n + m)$ time.*

We are now ready to describe an efficient offline procedure answering weighted ancestor queries.

Lemma 7.2. *Given a collection Q of weighted ancestor queries on a weighted tree T on n nodes with integer weights up to $(n + |Q|)^{O(1)}$, all queries from Q can be answered in $O(n + |Q|)$ time.*

Proof. We process the tree and the queries according to non-increasing weights. We maintain a union-find data structure that stores a partition of the set $V(T)$ of nodes of T. After the nodes with weight ℓ have been processed, each partition class is either a singleton of a node μ such that $\mathrm{weight}(\mu) < \ell$ or consists of the nodes of a subtree rooted at a node μ such that $\mathrm{weight}(\mu) \ge \ell$. In either case, μ is the identifier of the set.

Note that to update the partition for the next smaller value of ℓ, for each node μ with weight ℓ, it suffices to union the singleton $\{\mu\}$ with the node sets of subtrees rooted at the children of μ. Moreover, observe that after processing nodes at level ℓ, for each node ν, its ancestors μ with $\mathrm{weight}(\mu) < \ell$ form singletons, whereas ancestors μ with $\mathrm{weight}(\mu) \ge \ell$ belong to the same class as ν. Hence, the identifier of this class is the highest ancestor of μ with $\mathrm{weight}(\mu) \ge \ell$ (i.e., $\texttt{ancestor}(\nu, \ell) = \texttt{find}(\nu)$). Consequently, the weighted ancestor queries can be answered using Algorithm 2.

Next, we shall prove that Algorithm 2 can be implemented in $O(n + |Q|)$ time. Since node weights and query weights are integers bounded by $(n + |Q|)^{O(1)}$, they can be sorted using radix sort in $O(n + |Q|)$ time. For union-find operations, we need to identify nodes with integers $[1\,..\,n]$. We use the pre-order identifiers, as they guarantee that each partition class (which can be either a singleton or the node set of a subtree rooted at a given node) consists of consecutive integers. With $n - 1$ union operations and $|Q|$ find operations, Lemma 7.1 guarantees $O(n + |Q|)$ overall running time of the union-find data structure. □

8 IMPLEMENTATION OF THE OPERATION COMBINE

We describe here the missing part of the algorithm, which is based on computations on weighted trees. We first apply offline weighted ancestor queries to the suffix tree to obtain the following algorithmic tool.

Corollary 8.1. *Given a collection of subwords $s_1, \ldots, s_k$ of a word w of length n, each represented by an occurrence in w, in $O(n + k)$ total time we can compute the locus of each subword s_i in the suffix tree of w. Moreover, these loci can be made explicit in $O(n + k)$ extra time.*

ALGORITHM 2: Offline weighted ancestors for a weighted tree T and a set of queries Q

$W_T = \{\mathsf{weight}(\mu) : \mu \in V(T)\}$;
$W_Q = \{\ell : (v, \ell) \in Q\}$;
foreach $\ell \in W_T \cup W_Q$ *in the decreasing order* **do**
  foreach $\mu \in V(T) : \mathsf{weight}(\mu) = \ell$ **do**
    foreach v : *child of* μ *in the left-to-right order* **do**
      $\mathtt{union}(\mu, v)$;
  foreach $v : (v, \ell) \in Q$ **do**
    Report $\mathtt{ancestor}(v, \ell) := \mathtt{find}(v)$;

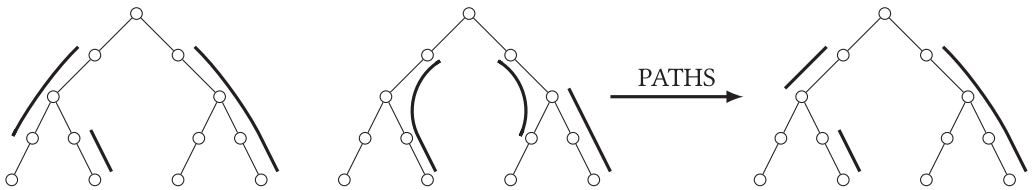

Fig. 11. The $\mathrm{PATHS}_T(P_1, P_2)$ operation: three copies of the same rooted tree T. The first two show P_1 and P_2, and the third one shows $\mathrm{PATHS}_T(P_1, P_2)$.

Proof. Let T be the suffix tree of w. Assume that $s_i = w[a\,..\,b]$ and consider the following nodes: the node μ representing $w[a\,..\,n]$ (the terminal node of T annotated with a) and $v = \mathtt{ancestor}(\mu, b - a + 1)$. If we denote by d the depth of v, then the locus of s_i is $(v, d - (b - a + 1))$. By Lemma 7.2, the loci of s_i can be computed in $O(n + k)$ time.

To make the corresponding implicit nodes explicit, we need to group them according to the nearest explicit descendant and sort them by distances from that node. This can be implemented in $O(n + k)$ time via radix sort. Then we simply create the explicit nodes in the obtained order. □

Let us introduce one more abstract operation on a rooted tree T. A *path family* is a family of pairwise disjoint paths in T, each leading downward. A path family is called *minimal* if there is no other path family covering the same set of nodes in T and consisting of a smaller number of paths.

Let $P_1, \ldots, P_m$ be path families in T. Then by $\mathrm{PATHS}_T(P_1, \ldots, P_m)$, we denote a minimal path family representing the nodes covered by all families $P_1, \ldots, P_m$ (see Figure 11).

Lemma 8.2. *The family* $\mathrm{PATHS}_T(P_1, \ldots, P_m)$ *can be computed in linear time with respect to the size of the tree* T*, the number* m*, and total number of paths in all families* P_i*.*

Proof. For each node μ in T, we would like to compute a value, denoted $S[\mu]$, that is equal to the number of paths in all families P_i that contain node μ. Observe that a node μ is covered by all families P_i if and only if $S[\mu] = m$.

For each node v in T, we will store a counter $C[v]$ so that, for a node μ, $S[\mu]$ is equal to the sum of values $C[v]$ across the nodes v in the subtree of μ. The counters C are initially set to zeroes. For each path leading from μ down to v in P_i, we increment the counter C at v and decrement the counter at the parent of μ (unless μ is the root). Next, for each node μ, we compute $S[\mu]$ as the sum of values $C[v]$ across the nodes v in the subtree of μ. This is done in a bottom-up fashion using the $S[\cdot]$ values of the children of μ.

This way, we compute all nodes represented by $\mathrm{PATHS}_T(P_1, \ldots, P_m)$. Finally, to find the minimal path family covering all of these nodes, we repeat the following process traversing the tree in a bottom-up order: If the value $S[\mu]$ is m, we create a single-node path $\{\mu\}$. If also $S[v] = m$ for a

child v of μ, we merge the paths containing these two nodes. (We choose the child arbitrarily if $S[v] = m$ for several children.) □

We are now ready to provide an efficient implementation of the Combine operation.

Lemma 8.3. *For a word w of length n and sets $R_1, \ldots, R_k$ of subwords of w, given in package representations of total size N, Combine$(R_1, \ldots, R_k)$ can be implemented in $O(n + N)$ time. The size of the resulting package representation is at most N.*

Proof. We reduce the problem to computing $\text{PATHS}_T(P_1, \ldots, P_m)$ for certain path families $P_1, \ldots, P_m$.

We first apply Corollary 8.1 for subwords $w[i \,..\, j_1]$ and $w[i \,..\, j_2]$ across all packages $\text{pack}(i, j_1, j_2)$ in $R_1, \ldots, R_m$, extending the set of explicit nodes of the suffix tree T of w by the obtained loci. For each package, we create a path connecting the corresponding two loci. The packages in a package representation are disjoint, so for each R_i this process results in a path family. We apply Lemma 8.2 to compute a minimal path family $P = \text{PATHS}_T(P_1, \ldots, P_m)$.

Finally, for each path in P, we create a package in T. We assume that each node stores the label ℓ of any terminal node in its subtree. If a path in P connects two nodes at depths $d_1 \le d_2$, and the second one stores the label ℓ, then we create a package $\text{pack}(\ell, \ell + d_1 - 1, \ell + d_2 - 1)$. □

9 CONCLUSION

We presented a linear-time algorithm that computes a representation of all seeds in a word. The representation that we proposed is simpler than the previous ones, as it uses only the suffix tree of the word and not the suffix tree of its reversal. Our algorithm is considerably simpler than the one presented in the preliminary version of the article [37]. A recent preprint [19] provides implementations of the $O(n \log n)$-time algorithm by Iliopoulos et al. [33], of our algorithm, and of an $O(n \log n)$-time version of our algorithm that applies an approach using a balanced binary tree instead of the recursive procedure. An experimental evaluation of the three algorithms presented by Czaka and Radoszewski [19] suggests that the $O(n \log n)$-time version of our algorithm is superiorly in practice.

ACKNOWLEDGMENTS

The authors thank Patryk Czajka for suggesting a simplification of the algorithm in Section 3.2 (Lemma 3.6).

REFERENCES

[1] Hayam Alamro, Golnaz Badkobeh, Djamal Belazzougui, Costas S. Iliopoulos, and Simon J. Puglisi. 2019. Computing the antiperiod(s) of a string. In *30th Annual Symposium on Combinatorial Pattern Matching, CPM 2019*, N. Pisanti and S. P. Pissis (Eds.), Vol. 128. Leibniz International Proceedings in Informatics. Schloss Dagstuhl–Leibniz-Zentrum für Informatik, Dagstuhl, Germany, Article 32, 11 pages. DOI: https://doi.org/10.4230/LIPIcs.CPM.2019.32

[2] Mai Alzamel, Maxime Crochemore, Costas S. Iliopoulos, Tomasz Kociumaka, Jakub Radoszewski, Wojciech Rytter, Juliusz Straszyński, Tomasz Waleń, and Wiktor Zuba. 2019. Quasi-linear-time algorithm for longest common circular factor. In *30th Annual Symposium on Combinatorial Pattern Matching, CPM 2019*, N. Pisanti and S. P. Pissis (Eds.), Vol. 128. Leibniz International Proceedings in Informatics. Schloss Dagstuhl–Leibniz-Zentrum für Informatik, Dagstuhl, Germany, Article 25, 14 pages. DOI: https://doi.org/10.4230/LIPIcs.CPM.2019.25

[3] Amihood Amir, Gad M. Landau, Moshe Lewenstein, and Dina Sokol. 2007. Dynamic text and static pattern matching. *ACM Transactions on Algorithms* 3, 2 (2007), 19. DOI: https://doi.org/10.1145/1240233.1240242

[4] Amihood Amir, Avivit Levy, Moshe Lewenstein, Ronit Lubin, and Benny Porat. 2019. Can we recover the cover? *Algorithmica* 81, 7 (2019), 2857–2875. DOI: https://doi.org/10.1007/s00453-019-00559-8

[5] Amihood Amir, Avivit Levy, Ronit Lubin, and Ely Porat. 2019. Approximate cover of strings. *Theoretical Computer Science* 793 (2019), 59–69. DOI: https://doi.org/10.1016/j.tcs.2019.05.020

[6] Amihood Amir, Avivit Levy, and Ely Porat. 2018. Quasi-periodicity under mismatch errors. In *29th Annual Symposium on Combinatorial Pattern Matching, CPM 2018*, G. Navarro, D. Sankoff, and B. Zhu (Eds.), Vol. 105. Leibniz

International Proceedings in Informatics. Schloss Dagstuhl–Leibniz-Zentrum für Informatik, Dagstuhl, Germany, Article 4, 15 pages. DOI: https://doi.org/10.4230/LIPIcs.CPM.2018.4

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

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

[9] Carl Barton, Tomasz Kociumaka, Chang Liu, Solon P. Pissis, and Jakub Radoszewski. 2020. Indexing weighted sequences: Neat and efficient. *Information and Computation* 270 (2020), 104462. DOI: https://doi.org/10.1016/j.ic.2019.104462

[10] Omer Berkman, Costas S. Iliopoulos, and Kunsoo Park. 1995. The subtree max gap problem with application to parallel string covering. *Information and Computation* 123, 1 (1995), 127–137. DOI: https://doi.org/10.1006/inco.1995.1162

[11] Omer Berkman, Baruch Schieber, and Uzi Vishkin. 1993. Optimal doubly logarithmic parallel algorithms based on finding all nearest smaller values. *Journal of Algorithms* 14, 3 (1993), 344–370. DOI: https://doi.org/10.1006/jagm.1993.1018

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

[13] Gerth Stølting Brodal and Christian N. S. Pedersen. 2000. Finding maximal quasiperiodicities in strings. In *11th Annual Symposium on Combinatorial Pattern Matching, CPM 2000 (LNCS)*, Raffaele Giancarlo and David Sankoff (Eds.), Vol. 1848. Springer, 397–411. https://doi.org/10.1007/3-540-45123-4_33

[14] Manolis Christodoulakis, Costas S. Iliopoulos, Kunsoo Park, and Jeong Seop Sim. 2005. Approximate seeds of strings. *Journal of Automata, Languages and Combinatorics* 10, 5–6 (2005), 609–626.

[15] Michalis Christou, Maxime Crochemore, Costas S. Iliopoulos, Marcin Kubica, Solon P. Pissis, Jakub Radoszewski, Wojciech Rytter, Bartosz Szreder, and Tomasz Waleń. 2013. Efficient seed computation revisited. *Theoretical Computer Science* 483 (2013), 171–181. DOI: https://doi.org/10.1016/j.tcs.2011.12.078

[16] Richard Cole, Costas S. Iliopoulos, Manal Mohamed, William F. Smyth, and Lu Yang. 2005. The complexity of the minimum *k*-cover problem. *Journal of Automata, Languages and Combinatorics* 10, 5-6 (2005), 641–653.

[17] Maxime Crochemore, Christophe Hancart, and Thierry Lecroq. 2007. *Algorithms on Strings*. Cambridge University Press. DOI: https://doi.org/10.1017/cbo9780511546853

[18] Maxime Crochemore and Wojciech Rytter. 2003. *Jewels of Stringology*. World Scientific. DOI: https://doi.org/10.1142/4838

[19] Patryk Czajka and Jakub Radoszewski. 2019. Experimental evaluation of algorithms for computing quasiperiods. arxiv:1909.11336.

[20] Martin Farach and S. Muthukrishnan. 1996. Perfect hashing for strings: Formalization and algorithms. In *7th Annual Symposium on Combinatorial Pattern Matching, CPM 1996 (LNCS)*, Daniel S. Hirschberg and Eugene W. Myers (Eds.), Vol. 1075. Springer, 130–140. https://doi.org/10.1007/3-540-61258-0_11

[21] Martin Farach-Colton, Paolo Ferragina, and S. Muthukrishnan. 2000. On the sorting-complexity of suffix tree construction. *Journal of the ACM* 47, 6 (2000), 987–1011. DOI: https://doi.org/10.1145/355541.355547

[22] Nathan J. Fine and Herbert S. Wilf. 1965. Uniqueness theorems for periodic functions. *Proceedings of the American Mathematical Society* 16, 1 (1965), 109–114. DOI: https://doi.org/10.2307/2034009

[23] Tomás Flouri, Costas S. Iliopoulos, Tomasz Kociumaka, Solon P. Pissis, Simon J. Puglisi, William F. Smyth, and Wojciech Tyczyński. 2013. Enhanced string covering. *Theoretical Computer Science* 506 (2013), 102–114. DOI: https://doi.org/10.1016/j.tcs.2013.08.013

[24] Harold N. Gabow and Roberd E. Tarjan. 1985. A linear-time algorithm for a special case of disjoint set union. *Journal of Computer and System Sciences* 30, 2 (1985), 209–221. DOI: https://doi.org/10.1016/0022-0000(85)90014-5

[25] Paweł Gawrychowski, Moshe Lewenstein, and Patrick K. Nicholson. 2014. Weighted ancestors in suffix trees. In *22th Annual European Symposium on Algorithms, ESA 2014 (LNCS)*, Andreas S. Schulz and Dorothea Wagner (Eds.), Vol. 8737. Springer, 455–466. https://doi.org/10.1007/978-3-662-44777-2_38

[26] Qing Guo, Hui Zhang, and Costas S. Iliopoulos. 2006. Computing the λ-seeds of a string. In *2nd International Conference on Algorithmic Aspects in Information and Management, AAIM 2006 (LNCS)*, Siu-Wing Cheng and Chung Keung Poon (Eds.), Vol. 4041. Springer, 303–313. https://doi.org/10.1007/11775096_28

[27] Qing Guo, Hui Zhang, and Costas S. Iliopoulos. 2006. Computing the minimum approximate λ-cover of a string. In *13th International Conference on String Processing and Information Retrieval, SPIRE 2006 (LNCS)*, Fabio Crestani, Paolo Ferragina, and Mark Sanderson (Eds.), Vol. 4209. Springer, 49–60. https://doi.org/10.1007/11880561_5

[28] Qing Guo, Hui Zhang, and Costas S. Iliopoulos. 2007. Computing the λ-covers of a string. *Information Sciences* 177, 19 (2007), 3957–3967. DOI: https://doi.org/10.1016/j.ins.2007.02.020

[29] Ondřej Guth. 2019. On approximate enhanced covers under Hamming distance. *Discrete Applied Mathematics* 274 (2019), 67–80. DOI: https://doi.org/10.1016/j.dam.2019.01.015

[30] Ondřej Guth, Borivoj Melichar, and Miroslav Balík. 2008. Searching all approximate covers and their distance using finite automata. In *Proceedings of the 20th Conference on Theory and Practice of Information Technologies (ITAT'08)*. http://ceur-ws.org/Vol-414/paper4.pdf.

[31] Lucian Ilie, Sheng Yu, and Kaizhong Zhang. 2002. Repetition complexity of words. In *8th Annual International Conference on Computing and Combinatorics, COCOON 2002 (LNCS)*, Oscar H. Ibarra and Louxin Zhang (Eds.), Vol. 2387. Springer, 320–329. https://doi.org/10.1007/3-540-45655-4_35

[32] Costas S. Iliopoulos, Manal Mohamed, and William F. Smyth. 2011. New complexity results for the *k*-covers problem. *Information Sciences* 181, 12 (2011), 2571–2575. DOI: https://doi.org/10.1016/j.ins.2011.02.009

[33] Costas S. Iliopoulos, Dennis W. G. Moore, and Kunsoo Park. 1996. Covering a string. *Algorithmica* 16, 3 (1996), 288–297. DOI: https://doi.org/10.1007/BF01955677

[34] Costas S. Iliopoulos and Laurent Mouchard. 1999. Quasiperiodicity: From detection to normal forms. *Journal of Automata, Languages and Combinatorics* 4, 3 (1999), 213–228.

[35] Costas S. Iliopoulos and William F. Smyth. 1998. An on-line algorithm of computing a minimum set of *k*-covers of a string. In *Proceedings of the 9th Australasian Workshop on Combinatorial Algorithms (AWOCA'98)*. 97–106.

[36] Juha Kärkkäinen, Peter Sanders, and Stefan Burkhardt. 2006. Linear work suffix array construction. *Journal of the ACM* 53, 6 (2006), 918–936. DOI: https://doi.org/10.1145/1217856.1217858

[37] Tomasz Kociumaka, Marcin Kubica, Jakub Radoszewski, Wojciech Rytter, and Tomasz Waleń. 2012. A linear time algorithm for seeds computation. In *23rd Annual ACM-SIAM Symposium on Discrete Algorithms (SODA'12)*, Yuval Rabani (Ed.). SIAM, 1095–1112. https://doi.org/10.1137/1.9781611973099

[38] Tomasz Kociumaka, Gonzalo Navarro, and Nicola Prezza. 2019. Towards a definitive measure of repetitiveness. arxiv:1910.02151.

[39] Tomasz Kociumaka, Solon P. Pissis, Jakub Radoszewski, Wojciech Rytter, and Tomasz Waleń. 2018. Efficient algorithms for shortest partial seeds in words. *Theoretical Computer Science* 710 (2018), 139–147. DOI: https://doi.org/10.1016/j.tcs.2016.11.035

[40] Tomasz Kociumaka, Solon P. Pissis, Jakub Radoszewski, Wojciech Rytter, and Tomasz Waleń. 2015. Fast algorithm for partial covers in words. *Algorithmica* 73, 1 (2015), 217–233. DOI: https://doi.org/10.1007/s00453-014-9915-3

[41] Roman M. Kolpakov and Gregory Kucherov. 1999. Finding maximal repetitions in a word in linear time. In *Proceedings of the 40th Annual Symposium on Foundations of Computer Science (FOCS'99)*. IEEE, Los Alamitos, CA, 596–604. DOI: https://doi.org/10.1109/SFFCS.1999.814634

[42] Yin Li and William F. Smyth. 2002. Computing the cover array in linear time. *Algorithmica* 32, 1 (2002), 95–106. DOI: https://doi.org/10.1007/s00453-001-0062-2

[43] M. Lothaire. 2002. *Algebraic Combinatorics on Words*. Encyclopedia of Mathematics and Its Applications (1st edition). Cambridge University Press. DOI: https://doi.org/10.1017/cbo9781107326019

[44] M. Lothaire. 2005. *Applied Combinatorics on Words*. Encyclopedia of Mathematics and Its Applications (1st edition). Cambridge University Press. DOI: https://doi.org/10.1017/cbo9781107341005

[45] Udi Manber and Eugene W. Myers. 1993. Suffix arrays: A new method for on-line string searches. *SIAM Journal on Computing* 22, 5 (1993), 935–948. DOI: https://doi.org/10.1137/0222058

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

[47] Dennis W. G. Moore and William F. Smyth. 1995. A correction to "An Optimal Algorithm to Compute All the Covers of a String". *Information Processing Letters* 54, 2 (1995), 101–103. DOI: https://doi.org/10.1016/0020-0190(94)00235-Q

[48] James H. Morris Jr. and Vaughan R. Pratt. 1970. *A Linear Pattern-Matching Algorithm*. Technical Report 40. Department of Computer Science, University of California, Berkeley.

[49] Sofya Raskhodnikova, Dana Ron, Ronitt Rubinfeld, and Adam D. Smith. 2013. Sublinear algorithms for approximating string compressibility. *Algorithmica* 65, 3 (2013), 685–709. DOI: https://doi.org/10.1007/s00453-012-9618-6

[50] Jeong Seop Sim, Kunsoo Park, Sung-Ryul Kim, and Jee-Soo Lee. 2002. Finding approximate covers of strings. *Journal of KIISE: Computer Systems and Theory* 29, 1 (2002), 16–21. http://www.koreascience.or.kr/article/ArticleFullRecord.jsp?cn=JBGHG6_2002_v29n1_16.

[51] William F. Smyth. 2000. Repetitive perhaps, but certainly not boring. *Theoretical Computer Science* 249, 2 (2000), 343–355. DOI: https://doi.org/10.1016/S0304-3975(00)00067-0

[52] Peter Weiner. 1973. Linear pattern matching algorithms. In *Proceedings of the 14th Annual Symposium on Switching and Automata Theory (SWAT'73)*. IEEE, Los Alamitos, CA, 1–11. DOI: https://doi.org/10.1109/SWAT.1973.13

Received March 2019; revised October 2019; accepted January 2020