A Linear Time Algorithm for Seeds Computation

Tomasz Kociumaka* Marcin Kubica* Jakub Radoszewski*† Wojciech Rytter*‡§
Tomasz Waleń*¶

Abstract

A seed in a word is a relaxed version of a period. We show a linear time algorithm computing a compact representation of all the seeds of a word, in particular, the shortest seed. Thus, we solve an open problem stated in the survey by Smyth (2000) and improve upon a previous over 15-year old $O(n \log n)$ algorithm by Iliopoulos, Moore and Park (1996). Our approach is based on combinatorial relations between seeds and a variant of the LZ-factorization (used here for the first time in context of seeds).

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, molecular biology etc. (see [26]). The concept of quasiperiodicity is a generalization of the notion of periodicity, and was defined by Apostolico & Ehrenfeucht in [1]. 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, while in a periodic repetition the occurrences of the period do not overlap. Hence, quasiperiodicity enables detecting repetitive structure of words when it cannot be found using the classical characterizations of periods. An extension of the notion of a cover is the notion of a seed — a cover which is not necessarily aligned with the ends of the word being covered, but is allowed to overflow on either side. Seeds were first introduced and studied by Iliopoulos, Moore and Park [20].

Covers and seeds have potential applications in DNA sequence analysis, namely in the search for regularities and common features in DNA sequences. The importance of the notions of quasiperiodicity follows also from their relation to text compression. Due to natural applications in molecular biology (a hybridization approach to analysis of a DNA sequence), both covers and seeds have also been extended in the sense that a number of factors are considered instead of a single word [22]. This way the notions of k-covers [8, 19], λ-covers [17] and λ-seeds [16] were introduced. In applications such as molecular biology and computer-assisted music analysis, finding exact repetitions is not always sufficient, the same problem holds for quasiperiodic repetitions. This leads to the introduction of the notions of approximate covers [29] and approximate seeds [6].

1.1 Previous results Iliopoulos, Moore and Park [20] gave an $O(n \log n)$ time algorithm computing a linear representation of all the seeds of a given word $w \in \Sigma^n$. For the next 15 years, no $o(n \log n)$ time algorithm was known for this problem. Computing all the seeds of a word in linear time was also set as an open problem in the survey [30]. A parallel algorithm computing all the seeds in $O(\log n)$ time and $O(n^{1+\varepsilon})$ (for any positive ε) space using n processors in the CRCW PRAM model was given by Berkman et al. [3]. An alternative sequential $O(n \log n)$ algorithm for computing the shortest seed was recently given by Christou et al. [7].

In contrast, a linear time algorithm finding the shortest cover of a word was given by Apostolico et al. [2] and later on improved into an on-line algorithm by Breslauer [4]. A linear time algorithm computing all the covers of a word was proposed by Moore & Smyth [28]. Afterwards an on-line algorithm for the all-covers problem was given by Li & Smyth [25].

Another line of research is finding maximal quasiperiodic subwords of a word. This notion resembles the maximal repetitions (runs) in a word [23], which is another widely studied notion of combinatorics on words. $O(n \log n)$ time algorithms for reporting all maximal quasiperiodic subwords of a word of length n have been proposed by Brodal & Pedersen [5] and Iliopoulos

*Faculty of Mathematics, Informatics and Mechanics, University of Warsaw, Warsaw, Poland, [kociumaka,kubica,jrad,rytter,walen]@mimuw.edu.pl

†Supported by the grant no. N206 568540 of the National Science Centre.

‡Faculty of Mathematics and Computer Science, Nicolaus Copernicus University, Toruń, Poland

§Supported by the grant no. N206 566740 of the National Science Centre.

¶Laboratory of Bioinformatics and Protein Engineering, International Institute of Molecular and Cell Biology in Warsaw, Poland

& Mouchard [21], these results improved upon the initial $O(n \log^2 n)$ time algorithm by Apostolico & Ehrenfeucht [1].

1.2 Our results We present a linear time algorithm computing a linear representation of all the seeds of a word. Such a representation is described in Section 11. This representation allows, among others, to find the shortest seed or a number of all the seeds in a very simple way. In the algorithm we assume that the alphabet Σ consists of integers, and its size is polynomial in terms of n. Hence, the letters of w can be sorted in linear time.

1.3 Our approach Our algorithm runs in linear time due to:

combinatorial properties of words: the Reduction-Property Lemma and Work-Skipping Lemma (formulated later on), which express the connection between seeds and factorizations; and

algorithmic results: linear time implementation of merging smaller results into larger, due to efficient processing of subtrees of a suffix tree (the function ExtractSubtrees), and efficient computation of long candidates for seeds (the function ComputeInRange).

2 Preliminaries

We consider *words* over Σ, $u \in \Sigma^*$; the positions in u are numbered from 1 to $|u|$. By Σ^n we denote the set of words of length n. For $u = u_1 u_2 \dots u_n$, let us denote by $u[i\,..\,j]$ a *subwords* of u equal to $u_i \dots u_j$.

2.1 Covers and seeds We say that a word v is a *cover* of w (covers w) if every letter of w is within some occurrence of v as a subword of w. A word v is a *seed* of w if it is a subword of w and w is a subword of some word u covered by v, see Fig. 1.

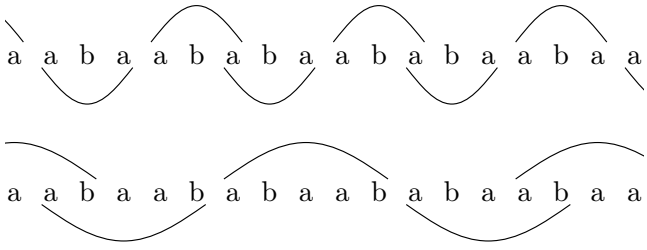

Figure 1: The word *aba* is the shortest seed of w = *aabaababaabababaabaa*. Another seed of w is *abaab*. In total, the word w has 35 distinct seeds.

2.2 Quasiseeds and border seeds Assume v is a subword of w. If w can be decomposed into $w = xyz$, where $|x|, |z| < |v|$ and v is a cover of y, then we say that v is a *quasiseed* of w. On the other hand, if w can be decomposed into $w = xyz$, so that $|x|, |z| < |v|$, v is a border of y and a seed of xvz, then v is a *border seed* of w. There holds the following simple relation between seeds, quasiseeds and border seeds.

OBSERVATION 2.1. *Let v be a subword of w. The word v is a seed of w if and only if v is a quasiseed of w and v is a border seed of w.*

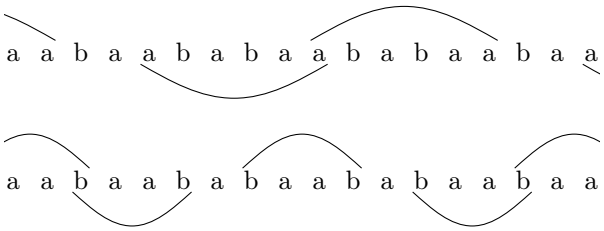

Figure 2: The word *ababaa* is a quasiseed of w = *aabaababaabababaabaa*, while the word *baab* is a border seed of w. Both are not seeds though. In total, the word w has 66 quasiseeds and 43 border seeds.

A *quasiseed* is a weaker and computationally easier version of the seed. The set of quasiseeds can be represented in a simple way on the suffix tree.

2.3 Quasigaps — quasiseeds on a suffix tree The *suffix tree* of the word w, denoted as T, is a compact TRIE of all suffixes of the word $w\#$, for $\# \notin \Sigma$ being a special end-marker. Recall that a suffix tree can be constructed in $O(n)$ time directly [14] or using the suffix array [10, 12].

For simplicity, we identify the nodes of T (both explicit and implicit) with the subwords of w which they represent. Leaves of T correspond to suffixes of w; the leaf corresponding to $w[i\,..\,n]$ is annotated with i. Let us introduce an equivalence relation on subwords of w. We say that two words are equivalent if the sets of start positions of their occurrences as subwords of w are equal. Note that this relation is very closely bonded with the suffix tree. Namely, each equivalence class corresponds to the set of implicit nodes lying on the same edge together with the explicit lower end of the edge. Quasiseeds belonging to the same equivalence class turn out to have a simple structure. To describe it, let us introduce the notion of a *quasigap*, a key notion for our algorithm.

DEFINITION 2.1. *Let v be a subword of w. If v is a quasiseed of w, then by* quasigap(v) *we denote the length*

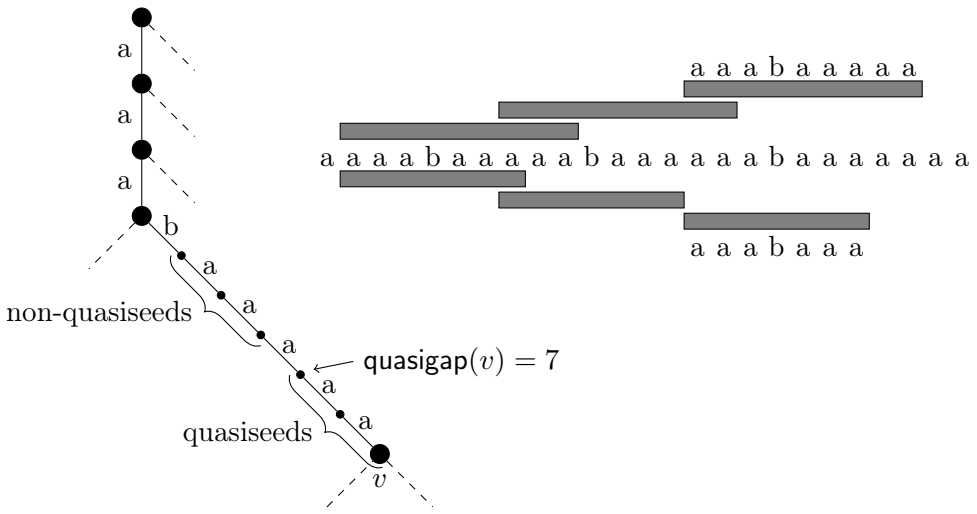

Figure 3: An illustration of quasigap(v) for the subword $v = aaabaaaaa$ of $w = aaaabaaaaabaaaaaabaaaaaaa$. Large dots represent explicit nodes and small dots represent implicit nodes of the suffix tree.

of the shortest quasiseed of w equivalent to v. Otherwise we define quasigap(v) *as* ∞.

The following observation provides a characterization of *all* quasiseeds of a given word using only the quasigaps of explicit nodes in the suffix tree T.

OBSERVATION 2.2. *All quasiseeds in a given equivalence class are exactly the prefixes of the longest word v in this class (corresponding to an explicit node in the suffix tree) of length at least* quasigap(v).

Example. Consider the words:

$$w = aaaabaaaaabaaaaaabaaaaaaa, \quad v = aaabaaaaa.$$

The equivalence class of v (denoted as $[v]$) is:

$$\{aaabaaaaa, aaabaaaa, aaabaaa, aaabaa, aaaba, aaab\}.$$

In $[v]$ only $aaabaaaaa$, $aaabaaaa$ and $aaabaaa$ are quasiseeds (see Fig. 3). In other words, quasiseeds in $[v]$ are the prefixes of v of length at least 7. Hence quasigap(v) = 7. Among these quasiseeds only $aaabaaaaa$ and $aaabaaaa$ are border seeds of w, and hence seeds of w.

3 An overview of the paper

In this section we describe informally the contents of the following sections and the relations between them.

The algorithm computing all the seeds of $w \in \Sigma^n$ is recursive. To be able to perform recursive computations, we extend the notion of quasigaps. For an interval $\gamma \subseteq [1\,..\,n]$, $\gamma = [i\,..\,j]$, we define γ-quasigaps as quasigaps in the subtree of the suffix tree T induced by the leafs from γ. A formal definition and some properties of γ-quasigaps are presented in Section 6. In the algorithm we wish to compute $[1\,..\,n]$-quasigaps.

The computation for an interval γ is reduced to computations for small subintervals (working intervals). The first key point is that the total size of these subintervals is small (at most half of the length of the interval γ). This is guaranteed by the Reduction-Property Lemma from Section 5.

The second key point is that information from the working intervals suffices to obtain the result for γ. Here Lempel-Ziv compression comes as a useful tool. Section 4 contains several combinatorial relations between certain families of subintervals in w and factorizations of w. The main idea is to divide the word into factors $f_1, \ldots, f_K$ (subword occurrences), so that any subword of a factor f_i also occurs in a concatenation of the earlier factors, i.e., in $f_1 \ldots f_{i-1}$. The working intervals are chosen from a family of short intervals covering γ (called later on a staircase family) as its elements which cross a border between factors. The number of borders between the factors corresponds to the number of the factors. Consequently we derive a relation between the number of factors in the factorization and the number of working intervals, stated formally as the Staircase-Factors Lemma in Section 4.

The main algorithm is presented in Section 7 and some of its parts are described in detail in Sections 8-10. In a single recursive call of the algorithm we divide the computed quasigaps into smaller and larger values, utilizing an observation formulated as the Work-Skipping Lemma in Section 5. To compute the smaller quasigap values, we make the recursive calls; in Sections 8, 9 we

show how to extract induced subtrees of a suffix tree and merge the information from these subtrees. In particular, in Section 9 we use a linear time algorithm for the (so called) Manhattan Skyline Problem extended to paths in a tree. We do not use recursion to compute the larger quasigaps, this is performed directly using the tools described in Section 10. This section is quite independent and its main part is an implementation of a linear time algorithm which computes all quasigaps in a range $[d, 2d]$ for some d. The algorithm is derived from simple combinatorics on words combined with simple algorithms for processing lists of small buckets.

The algorithm MAIN computes quasigaps, however the quasigaps are only secondary components of the solution. Our primary output information are the seeds. In Section 11 we show how to derive a representation of all seeds from the quasigaps. Here we mostly refer to previously known results.

In the last section we provide, with painful determination, the proof of a very technical fact (Lemma 9.1) related to correctness of merging information from subintervals and subtrees. This fact is intuitively not so difficult, unfortunately the use of induced subtrees makes the full proof technically and formally rather involved.

4 The tools

4.1 Factorizations An important tool used in the paper is the f-factorization which is a variant of the Lempel-Ziv factorization [31]. It plays an important role in optimization of text algorithms (see [9, 24, 27]). Intuitively, we can save some work, reusing the results computed for the previous occurrences of factors. We also apply this technique here. The f-factorization can be computed in linear time [11, 13], using so called *Longest Previous non-overlapping Factor* ($LPnF$) table.

From now on, let us fix a word $w \in \Sigma^n$. The intervals $[i\,..\,j] = \{i, i+1, \ldots, j\}$ will be denoted by small Greek letters. Assume all considered intervals satisfy $1 \le i \le j \le n$. We denote by $|\gamma|$ the length of the interval γ ($|[i\,..\,j]| \stackrel{\text{def}}{=} j - i + 1$).

By a *factorization* of a word w we mean a sequence $F = (f_1, f_2, \ldots, f_K)$ of factors of w such that $w = f_1 f_2 \ldots f_K$ and for each $i = 1, 2, \ldots, K$, f_i is a subword of $f_1 \ldots f_{i-1}$ or a single letter. Denote $|F| = K$. A factorization of w is called an *f-factorization* [10] if F has the minimal number of factors among all the factorizations of w. An f-factorization is constructed in a greedy way, as follows. Let $1 \le i \le K$ and $j = |f_1 f_2 \ldots f_{i-1}| + 1$. If $w[j]$ occurs in $f_1 f_2 \ldots f_{i-1}$, then f_i is the longest prefix of $w[j\,..\,|w|]$ that is a subword of $w[1\,..\,j-1]$. Otherwise $f_i = w[j]$.

4.2 Relation between seeds and factorizations There is a useful relation between covers and f-factorizations, which then extends for quasiseeds and thus seeds.

For a factorization F and interval λ, denote by F_λ the set of factors of F which lie completely within λ, that is start and end within λ.

LEMMA 4.1. *Let F be an f-factorization of w and $v \neq w$ be a cover of w. Then for $\lambda = [|v|+1\,..\,|w|]$ we have:*

$$|F_\lambda| \le \left\lfloor \tfrac{2\,|w|}{|v|} \right\rfloor - 1.$$

Before we proceed with the proof of the lemma, let us state the following claim.

CLAIM 4.1. *If v is a cover of w then there exists a sequence of at most $\left\lfloor \frac{2\,|w|}{|v|} \right\rfloor$ occurrences of v which completely covers w.*

Proof. The proof goes by induction over $|w|$. If $|w| \le 2\,|v|$ then the conclusion holds, since v is both a prefix and a suffix of w. Otherwise, let i be the starting position of the last occurrence of v in w such that $i \le |v|$. Now let j be the first position of the next occurrence of v in w. Note that both positions i, j exist and that $j > |v|$.

The word v is a cover of $w[j\,..\,|w|]$. By the inductive hypothesis, $w[j\,..\,|w|]$ can be covered by at most $\left\lfloor \frac{2(|w|-j+1)}{|v|} \right\rfloor \le \left\lfloor \frac{2\,|w|}{|v|} \right\rfloor - 2$ occurrences of v. Hence, w can be covered by all these occurrences together with those starting at 1 and at i. This concludes the inductive proof of the claim.

Proof of Lemma 4.1. By Claim 4.1, the word w can be completely covered with some occurrences of v at the positions $i_1, i_2, \ldots, i_p$, $1 = i_1 < i_2 < \ldots < i_p$, where $p \le \left\lfloor \frac{2\,|w|}{|v|} \right\rfloor$. Additionally define $i_{p+1} = |w| + 1$. Then there exists a factorization F' of w such that:

$$F'_\lambda = \{w[|v|+1\,..\,i_3-1]\} \quad \cup \quad \{w[i_j\,..\,i_{j+1}-1] \,:\, j = 3, 4, \ldots, p\}.$$

This set forms a factorization of $w[|v|+1\,..\,|w|]$ and consists of $p-1$ elements. This concludes that for any f-factorization F of w the set F_λ consists of at most $p-1$ elements, since otherwise we could substitute all the elements from this set by F'_λ, shorten the rightmost element of $F \setminus F_\lambda$ to the position $|v|$ and thus transform F into a factorization of w with a smaller number of factors, which is not possible.

As a corollary of Lemma 4.1, we obtain a similar fact for quasiseeds (hence, also for seeds).

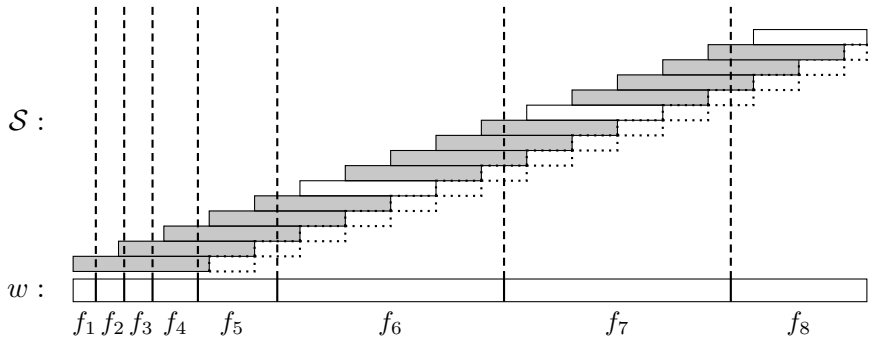

Figure 4: An m-staircase $\mathcal{S}$ of intervals covering w and an f-factorization $F = (f_1, f_2, \ldots, f_8)$ of w. The shaded intervals form the family $\mathcal{K} = \mathit{Reduce}(\mathcal{S}, F)$.

Lemma 4.2. (Quasiseeds-Factors Lemma)
Let F be an f-factorization of w and v be a quasiseed of w. Then for $\alpha = [2\ |v| + 1\,..\,|w| - |v|]$ we have:

$$|F_\alpha| \leq \left\lfloor \frac{2\,|w|}{|v|} \right\rfloor - 1.$$

4.3 Staircases and reduced staircases Another useful concept is that of a staircase of intervals. For a positive integer m, an interval staircase covering w is a sequence of intervals of length $3m$ with overlaps of size $2m$, starting at the beginning of w; possibly the last interval is shorter. More formally, for a given $1 \leq m \leq n$ an *m-staircase covering w* is a set of intervals covering w, defined as follows:

$$\mathcal{S} = \Big\{ [k \cdot m + 1\,..\,(k+3) \cdot m] \cap [1\,..\,n], \quad \text{for } k = 0, 1, \ldots, \max\Big(0, \Big\lceil \frac{n}{m} \Big\rceil - 3\Big) \Big\}.$$

Let $F = (f_1, f_2, \ldots, f_K)$ be an f-factorization of w. If $\mathcal{S}$ is an m-staircase covering w, let $\mathcal{K}$ be the family of those intervals $\lambda = [i\,..\,j] \in \mathcal{S}$, that $\lambda' = [i\,..\,\min(n, j+m)]$ (an extended interval) does not lie within a single factor of F, that is λ' overlaps more than one factor of F. Then we say that $\mathcal{K}$ is obtained by *a reduction of $\mathcal{S}$ with regard to the f-factorization F* and denote this by $\mathcal{K} = \mathit{Reduce}(\mathcal{S}, F)$.

There is a simple and useful relation between this reduction and the number of factors in a given factorization.

Lemma 4.3. (Staircase-Factors Lemma)
Assume we have an m-staircase $\mathcal{S}$ covering w and any factorization F of w. Then for an interval α:

$$|\{\lambda \in \mathit{Reduce}(\mathcal{S}, F) : \lambda \subseteq \alpha\}| \ \leq 4\,(|F_\alpha| + 1).$$

Proof. Each inter-position in w, in particular those between two consecutive factors of F, is covered by at most 4 extended intervals in a staircase.

5 Key lemmas

Finally, we are ready to prove two lemmas, both crucial for complexity analysis of the algorithm. The first one provides a limit for the total size of recursive calls, which are used to compute the small quasigaps, while the second allows to save much work while computing the large quasigaps.

The recursive calls are made for the intervals from $\mathit{Reduce}(\mathcal{S}, F)$. That is why we call them *working intervals*. For the whole algorithm to be linear it is necessary that the total length of those intervals is limited. The following lemma provides such a limit for a smart choice of the parameter m and for n which is large enough.

Lemma 5.1. (Reduction-Property Lemma)
Let $c = \frac{1}{50}$ and $n_0 = 200$ be constants. Assume that $|w| = n > n_0$, and let $\Delta = \lfloor cn \rfloor$. Let F be an f-factorization of w, and let $g = |F_{[2\Delta+1\,..\,n-\Delta]}|$; the factors from the latter set are called the middle factors *of F. Let $m = \left\lfloor \frac{cn}{g+1} \right\rfloor$. If $m > 0$ then for an m-staircase $\mathcal{S}$:*

$$\|\mathit{Reduce}(\mathcal{S}, F)\| < \tfrac{1}{2}n.$$

Here $\|\mathcal{J}\|$ denotes the total length of the intervals in a family $\mathcal{J}$.

Proof. Let us start with showing that $g \geq 3$, that is, that any f-factorization F has at least three middle factors. First of all, we have $\Delta \geq cn_0 > 0$.

Note that if a factor $f \in F$ starts at the position i in w, then $|f| < i$. Hence, the first factor in $F_{[2\Delta+1\,..\,n]}$ exists and starts at a position not exceeding 4Δ, and thus has the length at most 4Δ. Similarly, the second middle factor has the length at most 8Δ and the third has the length at most 16Δ. In total, the third considered factor ends at a position not exceeding 32Δ, and $32\Delta < n - \Delta$ by the choice of the parameter c.

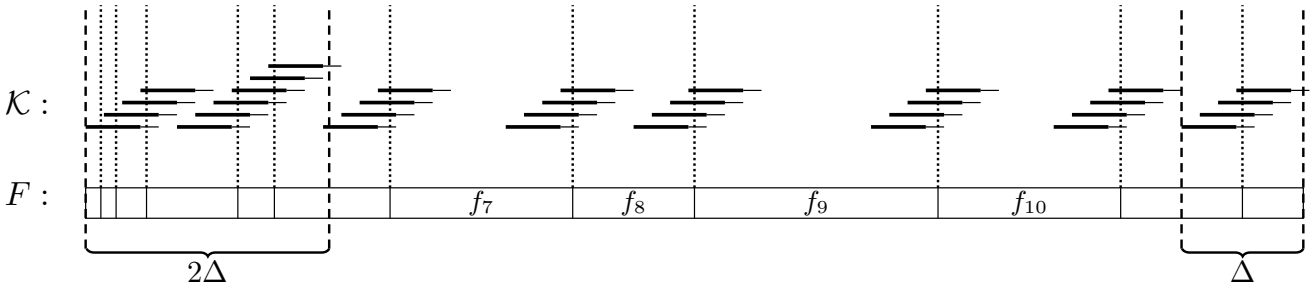

Figure 5: A word w of length n with an f-factorization $F = (f_1, f_2, \ldots, f_{12})$. Here the middle factors of F are: f_7, f_8, f_9, f_{10}, so $g = 4$. The segments above illustrate the family $\mathcal{K} = \mathit{Reduce}(\mathcal{S}, F)$. By the Reduction-Property Lemma, if these segments are short enough and Δ is small enough (yet linear in terms of n), then the total length of these intervals, $\|\mathcal{K}\|$, does not exceed $\frac{1}{2}n$.

This concludes that there must be at least three middle factors.

Now we proceed to the proof of the fact that $\|\mathcal{K}\| < \frac{1}{2}n$, where $\mathcal{K} = \mathit{Reduce}(\mathcal{S}, F)$. For this, we divide the intervals from $\mathcal{K}$ into three groups. First, let us consider intervals from $\mathcal{K}$ that start no later than at position 2Δ. Clearly, in $\mathcal{S}$ there are at most $\lceil \frac{2\Delta}{m} \rceil$ such intervals (recall that $m > 0$), therefore there are at most $\lceil \frac{2\Delta}{m} \rceil$ such intervals in the reduced staircase $\mathcal{K}$. Each of the intervals has the length at most $3m$, hence their total length does not exceed

$$3m \cdot \lceil \tfrac{2\Delta}{m} \rceil < 6\Delta + 3m \leq 6\Delta + 3 \lfloor \tfrac{cn}{4} \rfloor < 7 \cdot cn.$$

Now consider the intervals from $\mathcal{K}$ which end at some position $\geq n - \Delta + 1$. Each of the intervals starts at the position not smaller than:

$$n - \Delta + 1 - 3m + 1 \;\geq\; n - \Delta + 2 - \tfrac{3\Delta}{g+1} \;\geq\; n - \tfrac{7\Delta}{4} + 2.$$

In the second inequality we used the fact that $g \geq 3$. Hence, all the considered intervals from $\mathcal{K}$ are subintervals of an interval of length $\lceil \frac{7\Delta}{4} \rceil$. Similarly as in the previous case we obtain that their total length does not exceed

$$3m \cdot \lceil \tfrac{7\Delta}{4m} \rceil \leq \tfrac{21\Delta}{4} + 3m \leq \tfrac{21\Delta}{4} + 3 \lfloor \tfrac{cn}{4} \rfloor \leq 6 \cdot cn.$$

Finally, we consider the intervals from $\mathcal{K}$ which are subintervals of an interval

$$\alpha = [2\Delta + 1 \,..\, n - \Delta]\,.$$

Due to the Staircase-Factors Lemma, we have:

$$|\{\lambda \in \mathcal{K} : \lambda \subseteq \alpha\}| \;\leq\; 4(|F_\alpha| + 1) \;=\; 4(g+1).$$

The total length of such intervals does not exceed:

$$4 \cdot (g+1) \cdot 3m = 12(g+1) \left\lfloor \tfrac{cn}{g+1} \right\rfloor \;\leq\; 12 \cdot cn.$$

In conclusion, we have:

$$\|\mathcal{K}\| \;<\; 7 \cdot cn + 6 \cdot cn + 12 \cdot cn \;\leq\; \tfrac{1}{2}n.$$

A consequence of Lemma 4.2 is one of the key facts used to reduce the work of the algorithm. Due to this fact we can skip a significant part of the computations of large quasigaps.

Lemma 5.2. (Work-Skipping Lemma)
Let c, n_0 be as in Lemma 5.1. Let g be the number of middle factors in the f-factorization F of the word w, $w \in \Sigma^n$ for $n > n_0$. Then there is no quasiseed v of w such that:

$$\frac{2n}{g+1} \;<\; |v| \;\leq\; cn. \tag{5.1}$$

Proof. Assume to the contrary that there exists a quasiseed v which satisfies condition (5.1). By the Quasiseeds-Factors Lemma we obtain that

$$(|F_\alpha| + 1) \cdot |v| \leq 2\,|w|$$

where $\alpha = [2\,|v| + 1 \,..\, |w| - |v|]$. Due to the condition $|v| \leq cn = \Delta$, we have $|F_\alpha| \geq |F_{[2\Delta+1..n-\Delta]}| = g$. We conclude that $(g+1) \cdot |v| \leq 2\,|w|$, which contradicts the first inequality from (5.1).

6 A generalization to arbitrary intervals

The algorithm computing quasigaps (which represent quasiseeds) has a recursive structure. Unfortunately, the relation between quasiseeds in subwords of w and quasiseeds of entire w is quite subtle and we have to deal with technical representations of quasiseeds in the suffix tree. Even worse, the suffix trees of subwords of w are not exactly subtrees of the suffix tree of w. Due to these issues, our algorithm operates on subintervals of $[1 \,..\, n]$ rather than subwords of w. For an interval $\gamma =$

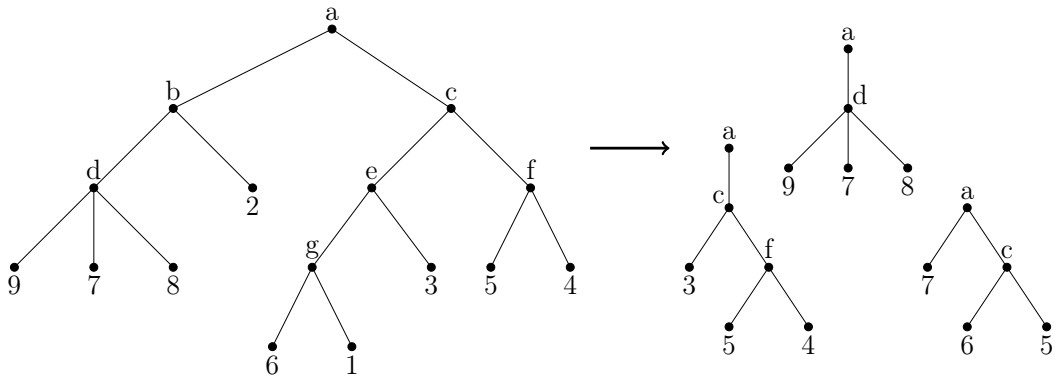

Figure 6: The induced subtree $T(\gamma)$ for $\gamma = [3\,..\,5]$, $\gamma = [7\,..\,9]$ and $\gamma = [5\,..\,7]$.

$[i\,..\,j]$, we can introduce the notions of f-factorization, interval staircase covering γ and reduced staircase in a natural way exactly as the corresponding notions for the subword $w[i\,..\,j]$. The remaining concepts require more attention.

6.1 Induced suffix trees Let us define an *induced suffix tree* $T(\gamma)$ for $\gamma = [i\,..\,j]$. It is obtained from T as follows. First, leaves labelled with numbers from γ and all their ancestors are selected, and all other nodes are removed. Then, the resulting tree is compacted, that is, all non-branching internal nodes become implicit. Still, the nodes (implicit and explicit) of such a tree can be identified with subwords of w. Of course, $T(\gamma)$ is *not* a suffix tree of $w[i\,..\,j]$.

By Nodes($T(\gamma)$) we denote the set of explicit nodes of $T(\gamma)$. Let v be a (possibly implicit) node of $T(\gamma)$. By $\mathit{parent}(v,\gamma)$ we denote the closest explicit ancestor of v. We assume that the root is the only node that is an ancestor of itself. Additionally, for an implicit node v of $T(\gamma)$, we define $\mathit{desc}(v,\gamma)$ as the closest explicit descendant of v. If v is explicit let $\mathit{desc}(v,\gamma) \stackrel{\text{def}}{=} v$.

6.2 Quasigaps in an arbitrary interval Here we extend the notion of quasigaps to a given interval γ. We provide definitions which are technically more complicated, however computationally more useful.

Before that we need to develop some notation. Let v be an arbitrary subword of w, or equivalently a node of T. By $\mathit{Occ}(v,\gamma)$ we denote the set of those starting positions of occurrences of v that lie within the interval γ. Note that the set $\mathit{Occ}(v,\gamma)$ represents the set of all leaves of $T(\gamma)$ in the subtree rooted at v. Let $\mathit{first}(v,\gamma) = \min \mathit{Occ}(v,\gamma)$ and $\mathit{last}(v,\gamma) = \max \mathit{Occ}(v,\gamma)$. Here we assume that $\min\emptyset = +\infty$, $\max\emptyset = -\infty$. A *maximum gap* of a set of integers $X = \{a_1, a_2, \ldots, a_k\}$, where $a_1 < a_2 < \ldots < a_k$, is defined as:

$$\mathsf{maxgap}(X) = \begin{cases} 0 & \text{if } |X| \le 1, \\ \max\{a_{i+1} - a_i : 1 \le i < k\} & \text{otherwise.} \end{cases}$$

For simplicity we abuse the notation and write $\mathsf{maxgap}(v,\gamma)$ instead of $\mathsf{maxgap}(\mathit{Occ}(v,\gamma))$. Now we are ready for the the main definition.

Definition 6.1. *Let v be a subword of w, and $\gamma = [i\,..\,j]$ be an interval. Denote $\ell_1 = \mathit{first}(v,\gamma) = \min \mathit{Occ}(v,\gamma)$, $\ell_2 = \mathit{last}(v,\gamma) = \max \mathit{Occ}(v,\gamma)$. Let:*

$$M = \max\Big(\mathit{maxgap}(v,\gamma),\ \ell_1 - i + 1,\ \left\lceil \tfrac{j-\ell_2}{2} \right\rceil + 1,\ |\mathit{parent}(v,\gamma)| + 1\Big). \quad (6.2)$$

If $M \le |v|$ then we define $\mathit{quasigap}(v,\gamma) = M$, otherwise $\mathit{quasigap}(v,\gamma) \stackrel{\text{def}}{=} \infty$.

If $\mathit{quasigap}(v,\gamma) \neq \infty$ then we call v a quasiseed in γ.

Observe that $\mathsf{quasigap}(v)$, defined in subsection 2.3, is exactly the same as $\mathsf{quasigap}(v, [1\,..\,|w|])$. In other words the quasiseeds of w are exactly the quasiseeds of w in $[1\,..\,|w|]$. Note that although both definitions are equivalent in this case, the former one is symmetric while the latter does not seem to be. That is because the occurrences are represented by their start position, and that representation is not symmetric.

For an arbitrary interval $[i\,..\,j]$ the quasiseeds of $w[i\,..\,j]$ are not necessarily quasiseeds of w in $[i\,..\,j]$. Intuitively, $\mathsf{quasigap}(v,\gamma)$ might not be equal to $\mathsf{quasigap}(v)$ restricted to the word $w[i\,..\,j]$, since the set $\mathit{Occ}(v,\gamma)$ may also depend on at most $|v|$ letters following γ. However, a careful but simple analysis of the definitions proves that the converse statement holds, i.e. the quasiseeds of $w[i\,..\,j]$ are quasiseeds of w in $[i\,..\,j]$. Hence, the Reduction-Property Lemma and the Work-Skipping Lemma hold for an arbitrary interval.

What is more, the quasiseeds in γ lying on a single edge of $T(\gamma)$ can still be described by a quasigap of the explicit lower end of the edge. More precisely, for $u = desc(v, \gamma)$, we have:

$$\mathsf{quasigap}(v,\gamma) = \begin{cases} \mathsf{quasigap}(u,\gamma) & \text{if } |v| \geq \mathsf{quasigap}(u,\gamma), \\ \infty & \text{otherwise.} \end{cases} \tag{6.3}$$

7 Main algorithm

In this section, we present a recursive algorithm for finding quasigaps. But before we go into details, let us roughly sketch the whole picture and give some intuitions.

7.1 An informal description of the algorithm

The problem of finding seeds can be reduced to finding quasiseeds (due to Observation 2.1) and checking which of them are also border seeds, what is described in Section 11. Moreover, the problem of finding quasiseeds can be, in turn, reduced to finding quasigaps, due to Observation 2.2.

The algorithm for calculating quasigaps for a given interval $\gamma \subseteq [1\,..\,n]$, $|\gamma| = N$, and a tree $T(\gamma)$ computes quasigaps of all explicit nodes of $T(\gamma)$. Let us call such nodes γ-*relevant*. It consists of two parts:

- The first part is non-recursive and works for *large* quasigaps (i.e., larger than m). This is done by the *working horse* of the algorithm — function *ComputeInRange*. This (rather simple) function is described in Section 10, and it does not depend on sections presenting other procedures.

- The second part works for the remaining quasigaps (called *small*) and is based on recursion. For a given interval γ the computation is reduced to computing *all* quasigaps for *working* subintervals $\lambda \subseteq \gamma$. These subintervals constitute the *reduced* staircase $\mathcal{K}$. Section 8 shows how all the trees $T(\lambda)$ can be extracted from $T(\gamma)$ in $O(N + \|\mathcal{K}\|)$ time.

  The results for working intervals are merged in $O(N + \|\mathcal{K}\|)$ time, by employing function Merge, described in Section 9.

  Due to Reduction-Property Lemma, the total length of the intervals in $\mathcal{K}$ is less than $\frac{1}{2}N$, what guarantees a linear total time complexity.

7.2 A detailed description

First, we compute an f-factorization F of the word $w[i\,..\,j]$. Let $\Delta = \lfloor \frac{N}{50} \rfloor$ and $m = \left\lfloor \frac{N}{50(g+1)} \right\rfloor$, where g is the number of middle factors of F. We divide the values of finite quasigaps of γ-relevant nodes into values exceeding m (*large* quasigaps) and the remaining not exceeding m (*small* quasigaps). Note that if $m = 0$ then *all* quasigaps of γ-relevant nodes are considered large.

Due to the Work-Skipping Lemma, the large quasigaps can belong to two ranges:

$$\left[m+1, \tfrac{2N}{g+1}\right] \cup [\Delta+1, N] \subseteq [m+1, 100(m+1)] \cup [\Delta+1, 50(\Delta+1)].$$

Thus all large quasigaps can be computed using the algorithm ComputeInRange(l, r, γ), described in Section 10. This algorithm computes all quasigaps of γ-relevant nodes which are in the range $[l, r]$ in $O(N(1 + \log \frac{r}{l}))$ time. We apply it for $r \leq 100l$, obtaining $O(N)$ time complexity.

If $m = 0$ there are no small quasigaps. Otherwise the small quasigaps are computed recursively as follows. We start by introducing an m-staircase $\mathcal{S}$ of intervals covering γ and reduce the staircase $\mathcal{S}$ with respect to F to obtain a family $\mathcal{K}$, see also Fig. 4. Recall that, by the Reduction-Property Lemma, $\|\mathcal{K}\| \leq N/2$. We compute the quasigaps for all intervals $\lambda \in \mathcal{K}$ recursively. However, before that, we need to create the trees $T(\lambda)$ for $\lambda \in \mathcal{K}$, which can be done in $O(N)$ total time using the procedure ExtractSubtrees(γ, $\mathcal{K}$), described in Section 8. Finally, the results of the recursive calls are put together to determine the small quasigaps in $T(\gamma)$. This Merge procedure is described in Section 9. Algorithm 1 briefly summarizes the structure of the MAIN algorithm.

THEOREM 7.1. *The algorithm MAIN works in $O(N)$ time, where $N = |\gamma|$.*

Proof. All computations performed in a single recursive call of MAIN work in $O(N)$ time. These are: computing the f-factorization in line 4 (see [11]), computing large quasigaps in lines 8–9 using the function ComputeInRange (see the Section 10), computing the family of working intervals $\mathcal{K}$ and the trees $T(\lambda)$ for $\lambda \in \mathcal{K}$ in lines 12–14 (see Lemma 8.1 in Section 8) and merging the results of the recursive calls in line 17 of the pseudocode (see Lemma 9.3 in Section 9). We perform recursive calls for all $\lambda \in \mathcal{K}$, the total length of the intervals from $\mathcal{K}$ is at most $N/2$ (by the Reduction-Property Lemma). We obtain the following recursive formula for the time complexity, where $M = |\mathcal{K}|$ and C is a constant:

$$Time(N) \leq C \cdot N + \sum_{i=1}^{M} Time(N_i), \quad \text{where } N_i > 0, \ \sum_{i=1}^{M} N_i \leq \tfrac{N}{2}.$$

Algorithm 1: Recursive procedure MAIN(γ)

Input: An interval $\gamma = [i\,..\,j]$ and the tree $T(\gamma)$ induced by suffixes starting in γ.
Output: quasigap(v, γ) for γ-relevant nodes of $T(\gamma)$.

if $|\gamma| \leq 200$ **then**
    ComputeInRange(1, $|\gamma|$, γ)
    return
$F :=$ f-factorization($w[i\,..\,j]$)
$\Delta := \left\lfloor \frac{|\gamma|}{50} \right\rfloor$
$g := \left|F_{[i+2\Delta, j-\Delta]}\right|$ `// the number of middle factors`
$m := \left\lfloor \frac{|\gamma|}{50(g+1)} \right\rfloor$
ComputeInRange($\Delta + 1$, $50(\Delta + 1)$, γ) `// large quasigaps`
ComputeInRange($m + 1$, $100(m + 1)$, γ) `// large quasigaps`
if $m = 0$ **then**
    return `// no small quasigaps, hence no recursive calls`
$\mathcal{S} :=$ IntervalStaircase(γ, m)
$\mathcal{K} :=$ Reduce($\mathcal{S}, F$) `// the total length of intervals in` $\mathcal{K}$ `is` $\leq |\gamma|/2$
ExtractSubtrees(γ, $\mathcal{K}$) `// prepare the trees` $T(\lambda)$ `for` $\lambda \in \mathcal{K}$
foreach $\lambda \in \mathcal{K}$ **do**
    MAIN(λ)
Merge(γ, $\mathcal{K}$) `// merge the results of the recursive calls`

This formula easily implies that $Time(N) \leq 2C \cdot N$.

In the following three sections we fill in the description of the algorithm by showing how to implement the functions ExtractSubtrees, Merge and ComputeInRange efficiently.

8 Implementation of ExtractSubtrees

In this section we show how to extract all subtrees $T(\lambda)$ from the tree $T(\gamma)$, for $\lambda \in \mathcal{K}$. Let us fix $\lambda \subseteq \gamma$. Assume that all integer elements from the interval λ are sorted in the order of the corresponding leaves of $T(\gamma)$ in a left-to-right traversal: $(i_1, \ldots, i_{|\lambda|})$. Then the tree $T(\lambda)$ can be extracted from $T(\gamma)$ in $O(|\lambda|)$ time using an approach analogous to the construction of a suffix tree from the suffix arrays [10, 12], see also Fig. 7. In the algorithm we maintain the rightmost path P of $T(\lambda)$, starting with a single edge from the leaf corresponding to i_1 to the root of $T(\gamma)$. For each i_j, $j = 2, \ldots, M$, we find the (implicit or explicit) node u of P of depth equal to the lowest common ancestor (LCA) of the leaves i_{j-1} and i_j in $T(\gamma)$, traversing P in a bottom-up manner. The node u is then made explicit, it is connected to the leaf i_j and the rightmost path P is thus altered. Recall that lowest common ancestor queries in $T(\gamma)$ can be answered in $O(1)$ time with $O(N)$ preprocessing time [18].

Thus, to obtain an $O(N)$ time algorithm computing all the trees $T(\lambda)$, it suffices to sort all the elements of each interval $\lambda \in \mathcal{K}$ in the left-to-right order of leaves of $T(\gamma)$. This can be done, using counting sort or bucket sort, for all the elements of $\mathcal{K}$ in $O(|\gamma| + \|\mathcal{K}\|) = O(N)$ total time. Thus we obtain the following result.

LEMMA 8.1. *ExtractSubtrees($\gamma, \mathcal{K}$) can be implemented in $O(|\gamma| + \|\mathcal{K}\|)$ time.*

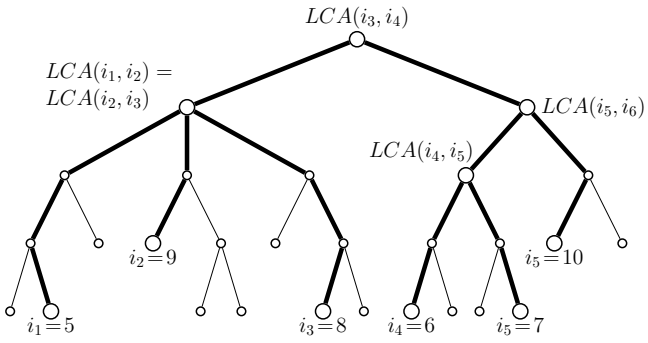

Figure 7: An illustration of the induced subtree $T(\lambda)$ for $\lambda = [5\,..\,10]$ extracted from $T(\gamma)$. The explicit nodes of $T(\lambda)$ are leaves $i_1, i_2, \ldots, i_6$ and the branching nodes $LCA(i_1, i_2), \ldots, LCA(i_5, i_6)$. Some of those LCA's may be equal.

9 Implementation of Merge

In this section we describe how to assemble results of the recursive calls of MAIN to determine the small quasigaps. Speaking more formally, we provide an algorithm that given an interval γ, a positive integer m, $\mathcal{K}$ – a reduced m-staircase of intervals covering γ and quasigaps of explicit nodes in $T(\lambda)$ for all $\lambda \in \mathcal{K}$, computes those quasigaps of explicit nodes in $T(\gamma)$, that are not larger than m. More precisely, for every explicit node v in $T(\gamma)$, the algorithm either finds the exact value of $\mathsf{quasigap}(v, \gamma)$ or says that it is larger than m. In the algorithm we use the following crucial lemma which provides a way of computing $\mathsf{quasigap}(v, \gamma)$ using the quasigaps of the nodes from $T(\lambda)$ for all $\lambda \in \mathcal{K}$, provided that $\mathsf{quasigap}(v, \gamma) \leq m$. Its rather long and complicated proof can be found in the last section of the paper.

LEMMA 9.1. *Assume $m > 0$. Let v be an explicit node in $T(\gamma)$.*
(a) If v is neither an explicit nor an implicit node in $T(\lambda)$ for some $\lambda \in \mathcal{K}$ then $\mathsf{quasigap}(v, \gamma) > m$.
(b) If $|parent(v, \gamma)| \geq m$ then $\mathsf{quasigap}(v, \gamma) > m$.
(c) If the conditions from (a) and (b) do not hold, let

$$M'(v) = \max\{\mathsf{quasigap}(desc(v, \lambda), \lambda) : \lambda \in \mathcal{K}\}.$$

If $M'(v) \leq \min(m, |v|)$ then $\mathsf{quasigap}(v, \gamma) = M'(v)$, otherwise $\mathsf{quasigap}(v, \gamma) > m$.

To obtain an efficient implementation of the criterion from Lemma 9.1, we utilize the following auxiliary problem. Note that the "max" part of this problem is a generalization of the famous Skyline Problem for trees.

PROBLEM 9.1. (TREE-PATH-PROBLEM) *Let T be a rooted tree with q nodes. By $P_{v,u}$ we denote the path from v to its ancestor u, excluding the node u. Let $\mathcal{P}$ be a family of paths of the form $P_{v,u}$, each represented as a pair (v, u). To each $P \in \mathcal{P}$ a weight $w(P)$ is assigned, we assume that $w(P)$ is an integer of size polynomial in n. For each node v of T compute $\max\{w(P) : v \in P\}$ and $\sum_{P:v\in P} w(P)$.*

LEMMA 9.2. *The Tree-Path-Problem can be solved in $O(q + |\mathcal{P}|)$ time.*

Proof. Consider first the "max" part of the Tree-Path-Problem, in which we are to compute, for each $v \in \text{Nodes}(T)$, the maximum of the w-values for all paths $P \in \mathcal{P}$ containing v (denote this value by $W_{\max}(v)$). We will show a reduction of this problem to a restricted version of the find/union problem, in which the structure of the union operations forms a static (known in advance) tree. This problem can be solved in linear time [15].

We will be processing all elements of $\mathcal{P}$ in the order of non-increasing values of w. We store a partition of the set Nodes(T) into disjoint set, which are *connected*, that is if two nodes are in a single set then all nodes lying on the only path connecting them also belong to that set. Such sets are represented by their topmost nodes (note that each set has a unique node of lowest level). Initially each node forms a singleton set. Throughout the algorithm we will be assigning $W_{\max}$ values to the nodes of T, maintaining an invariant that nodes without an assigned value form single-element sets.

When processing a path $P_{v,u} \in \mathcal{P}$, we identify all the sets $S_1, S_2, \ldots, S_k$ which intersect the path. For this, we start in the node v, go to the root of the set containing v, proceed to its parent, and so on, until we reach the set containing the node u. For all singleton sets among S_i, we set the value $W_{\max}$ of the corresponding node to $w(P_{v,u})$, provided that this node was not assigned the $W_{\max}$ value yet. Finally, we union all the sets S_i.

Note that the structure of the union operations is determined by the child-parent relations in the tree T, which is known in advance. Thus all the find/union operations can be performed in linear time [15], which yields $O(q + |\mathcal{P}|)$ time in total.

Now we proceed to an implementation of the "+" part of the Tree-Path-Problem (we compute the values $W_+(v)$). This time, instead of considering a path $P_{v,u}$ with the value $w(P_{v,u})$, we consider a path $P_{v,root}$ with the value $w(P_{v,u})$ and a path $P_{u,root}$ with the value $-w(P_{v,u})$. Now each considered path leads from a node of T to the root of T.

For each $u \in \text{Nodes}(T)$ we store, as $W'(u)$, the sum of weights of all such paths starting in u. Note that $W_+(v)$ equals the sum of $W'(u)$ for all u in the subtree rooted at v. Hence, all W_+ values can be computed in $O(n)$ time by a simple bottom-up traversal of T.

Now let us explain how the implementation of Merge can be reduced to the Tree-Path-Problem. In our case $T = T(\gamma)$. Observe that for each $\lambda \in \mathcal{K}$ an edge from the node u down to the node v in $T(\lambda)$ induces a path $P_{v,u}$ in $T(\gamma)$. Let $\mathcal{P}$ be a family of all such edges. If we set the weight of each path $P_{v,u}$ corresponding to an edge (u, v) in $T(\lambda)$ to 1, we can identify all nodes v' which are either explicit or implicit in each $T(\lambda)$ for $\lambda \in \mathcal{K}$ as exactly those nodes for which the sum of the corresponding $w(P)$ values equals $|\mathcal{K}|$. On the other hand, if we set the weight of such path to $\mathsf{quasigap}(v, \lambda)$, we can compute for each v' explicit in $T(\gamma)$ the maximum of $\mathsf{quasigap}(desc(v', \lambda), \lambda)$ over such λ that v' is an explicit or implicit node in $T(\lambda)$. In particular for nodes

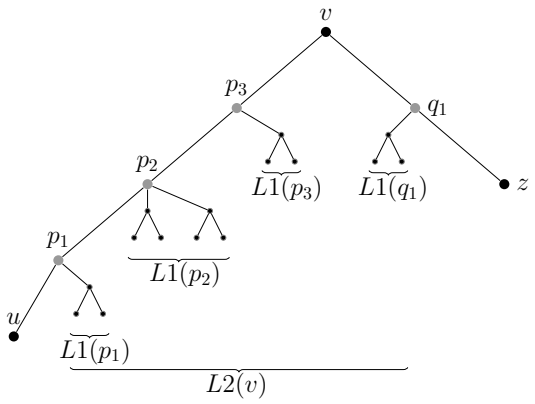

Figure 8: A sample tree with easy nodes marked in black and hard nodes marked in grey.

identified in the previous part, this maximum equals $M'(v')$. All the remaining conditions from Lemma 9.1 can trivially be checked in time proportional to the size of $T(\gamma)$, that is $O(|\gamma|)$. Note that $|\mathcal{P}| = O(\|\mathcal{K}\|)$ and all weights are from the set $[0\,..\,|\gamma|] \cup \{\infty\}$, and therefore can be treated as bounded integers. Thus, as a consequence of Lemma 9.2, we obtain the following corollary.

LEMMA 9.3. *Merge$(\gamma, \mathcal{K})$ can be implemented in $O(|\gamma| + \|\mathcal{K}\|)$ time.*

10 Implementation of ComputeInRange

In this section we show how to compute quasigaps in a range $[l, r]$ for γ-relevant nodes of $T(\gamma)$. More precisely, for each $v \in \text{Nodes}(T(\gamma))$ we either compute the exact value of $\mathsf{quasigap}(v, \gamma)$, report that $\mathsf{quasigap}(v, \gamma) < l$ or that $\mathsf{quasigap}(v, \gamma) > r$, we call such values $[l, r]$-*restricted quasigaps*. Note that if we made the range larger, we still solve the initial problem, hence we may w.l.o.g. assume that $\frac{r}{l}$ is a power of 2. This lets us split the range $[l, r]$ into several ranges of the form $[d, 2d]$. Below we give an algorithm for such intervals running in $O(N)$ time (where $N = |\gamma|$), which for an arbitrary range gives $O(N(1 + \log \frac{r}{l}))$ time complexity.

10.1 The structure of the algorithm For many nodes v we can easily check that $\mathsf{quasigap}(v) > 2d$. Thus, we may limit ourselves to the nodes which we call *active*, that is, such $v \in \text{Nodes}(T(\gamma))$ that the following conditions hold:

$$\begin{aligned} |Occ(v, \gamma)| &\geq \tfrac{N}{2d}, \\ first(v, \gamma) &< i + 2d, \\ last(v, \gamma) &> j - 4d + 1. \end{aligned} \tag{10.4}$$

If v is not an active node then certainly $\mathsf{quasigap}(v, \gamma) > 2d$.

Observe that if v is active then $parent(v, \gamma)$ is also active. Hence, active nodes induce a subtree of T. Let us call the branching nodes of this tree *easy*. More precisely, the easy nodes are: the root of $T(\gamma)$ and the active nodes which have either none or more than one active child nodes. The remaining active nodes are called *hard* nodes. Obviously, all easy and hard nodes of $T(\gamma)$ can be found in $O(N)$ time. The following lemma shows that the number of easy active nodes is very limited.

LEMMA 10.1. *The tree $T(\gamma)$ contains $O(d)$ easy nodes.*

Proof. Let us divide the set of all easy nodes of $T(\gamma)$ into the easy nodes having active children (the set X) and all the remaining easy nodes (the set Y). Apart from at most one node (the root of $T(\gamma)$), each node from X has at least two child subtrees containing easy nodes, hence $|X| \leq |Y|$. The size of the subtree of $T(\gamma)$ rooted at any node from the set Y is $\Omega(N/d)$ and all such subtrees are pairwise disjoint. Hence, $|Y| = O(d)$ and $|X| + |Y| = O(d)$.

Quasigaps of active nodes will be computed using (6.2). Note that the only computationally difficult part of this equation is the maxgap. However, the $[d, 2d]$-restricted maxgaps for all active nodes can still be found in linear time. Precisely, for an active node v, we either find the exact $\mathsf{maxgap}(v, \gamma)$ or report that $\mathsf{maxgap}(v, \gamma) < d$ or that $\mathsf{maxgap}(v, \gamma) > 2d$.

The main idea of the algorithm computing restricted maxgaps is to have a bucket for each d consecutive elements of γ (with the last bucket possibly smaller). Note that, since the gaps between elements of the same bucket are certainly smaller than d, the $[d, 2d]$-restricted $\mathsf{maxgap}(v, \gamma)$ depends only on the first and the last element of each bucket. Due to the small number of easy nodes, this observation on its own lets

Algorithm 2: Computing restricted maxgap values for easy nodes

Input: A suffix tree $T(\gamma)$, and an integer value d.
Output: The suffix tree with easy nodes annotated with maxgap values, for some nodes we can use labels "$< d$" or "$> 2d$".

1 **for** $v \in$ *easy-nodes*$(T(\gamma))$ *(bottom to the top)* **do**
2 Initialize B_v with empty doubly-linked lists
3 **foreach** $u \in$ *easy-desc*(v) **do**
4 UpdateBuckets(B_v, B_u)
5 UpdateBuckets(B_v, $L2(v)$)
6 Replace each $B_v[k]$ of size at least 3 with $\{head(B_v[k]), tail(B_v[k])\}$
7 Compute maxgap(v) using the contents of B_v, use labels "$< d$" or "$> 2d$" if the result is outside the range $[d, 2d]$

Algorithm 3: UpdateBuckets(B, L) procedure

Input: An array of buckets B and a list of occurrences L.
Output: The buckets from B updated with the positions from L.

1 **foreach** $l \in L$ **do**
2 Let k be the bucket assigned to the position l
3 **if** $empty(B[k])$ **then** $B[k] := \{l\}$
4 **else if** $l < head(B[k])$ **then**
5 add l to the front of $B[k]$
6 **else if** $l > tail(B[k])$ **then**
7 add l to the back of $B[k]$
8 **else** we can ignore l

us compute $[d, 2d]$-restricted maxgaps of all easy nodes in $O(N)$ time. The computation for hard nodes requires more attention. We use the fact that all such nodes can be divided into $O(d)$ disjoint paths connecting pairs of easy nodes to develop an algorithm for more efficient bucket processing.

For each active node we define the list $L1(v)$ which consists of all $l \in Occ(v, \gamma)$ such that the path from v to l contains only one active node (namely v). For each easy node also we define the list $L2(v)$ which consists of all $l \in Occ(v, \gamma)$ such that the path from v to l contains only one easy node (again, the node v). A sample tree with easy and active nodes along with both kinds of lists marked is presented in Fig. 8. Note that each leaf can be present in at most one list $L1$, and at most one list $L2$, and, additionally, that all the lists $L1$ and $L2$ can be constructed in $O(N)$ total time by a simple bottom-up traversal of $T(\gamma)$. For each easy node we introduce the set easy-desc(v) (immediate easy descendants) which consists of all easy descendants u of v such that the path from v to u contains no easy nodes apart from v and u themselves. The algorithm first computes the $[d, 2d]$-restricted maxgaps of all easy nodes, and later on for all the hard nodes.

10.2 Processing easy nodes For each easy node v, Algorithm 2 computes an array $B_v[0\,..\,\lceil n/d \rceil]$, such that $B_v[k]$ contains the minimal and the maximal elements in the set $Occ(v, [i+d\cdot k\,..\,i+d\cdot(k+1)-1])$, provided that this set is not empty. To fill the B_v array, the algorithm inspects all elements of $L2(v)$ and the arrays B_u for all $u \in$ easy-desc(v). For this, an auxiliary procedure UpdateBuckets(B, L) is utilized, in which, while being updated, each bucket B_v always contains an increasing sequence of elements. Afterwards, the $[d, 2d]$-restricted maxgap of v can be computed by a single traversal of the B_v array.

Claim 10.1. *Algorithm 2 has time complexity $O(N)$.*

Proof. The total size of all the arrays B_v is $O(N)$, since there are $O(d)$ easy nodes and each such array has $O(N/d)$ elements, each of constant size. We only need to investigate the total time of all UpdateBuckets(B_v, L) calls (note that a single call works in $O(|L|)$ time). Recall that the total length of all the lists $L2$ is $O(N)$, therefore the calls in line 5 of the algorithm take $O(N)$ total time. Similarly, each array B_u is used at most once in a call of UpdateBuckets(B_v, B_u) from line 4, so this step also takes $O(N)$ time in total.

10.3 Processing hard nodes Algorithm 4 processes hard nodes in paths $p_1, \ldots, p_a$, such that $p_0 = v$ and $parent(p_a)$ are easy nodes and $\forall_k p_k = parent(p_{k-1})$. Here we also store an array of buckets B, but this time each $B[k]$ may contain more than 2 elements (stored in a doubly-linked list). Starting from B_v, we update B using the lists $L1(p_1), \ldots, L1(p_a)$. The $[d, 2d]$-restricted maxgap for p_a is computed, by definition, from the array B, but the computations for $p_{a-1}, \ldots, p_1$ require more attention. This time we remove elements of the lists $L1(p_a), \ldots, L1(p_1)$ from the buckets B. If we remove an occurrence l corresponding to bucket $B[q]$, the $[d, 2d]$-restricted maxgap value can be altered only if l is the head or the tail of the list $B[q]$. Otherwise the list in $B[q]$ does not even contain ℓ (since we process $L1$ backwards this time). In this case, if l is neither the globally first nor the globally last occurrence in B, the $[d, 2d]$-restricted maxgap can only increase, and we can

Algorithm 4: Computing restricted maxgap values for active nodes (including hard nodes)

Input: A suffix tree $T(\gamma)$, $\gamma = [i\,..\,j]$, and an integer value d.
Output: The suffix tree with all active nodes annotated with $[d, 2d]$-restricted maxgap.

```
1  Compute buckets B_v and maxgap for all easy nodes using Algorithm 2
2  foreach v ∈ easy-nodes(T(γ)) do
3      Let p_1, ..., p_a be the maximal path of hard nodes defined as p_0 = v
4      p_k := parent(p_{k-1}, γ)
5      curr := maxgap(v); B := B_v
6      for k = 1 to a do UpdateBuckets(B, L1(p_k))
7      for k = a downto 1 do
8          maxgap(p_k) := curr
9          foreach l ∈ L1(p_k) (backwards) do
10             let q be the bucket assigned to the occurrence l
11             if l = head(B[q]) or l = tail(B[q]) then
12                 remove l from B[q]
13                 if l < i + 2d or l > j − 4d + 1 then
14                     curr := recompute the maxgap from the buckets B
15                 else
16                     curr := update the maxgap in curr using B[q − 2 .. q + 2]
17 Replace maxgap values outside range [d, 2d] with labels "< d" or "> 2d"
```

update it in constant time by investigating neighboring buckets $B[q-2..q+2]$. Otherwise we recompute the $[d, 2d]$-restricted maxgap using all buckets. Note that due to the restriction on *first* and *last* in the definition of active nodes (10.4), the latter case may happen only if we delete something from one of the first few or the last few buckets, hence this case is handled in line 14 of the algorithm.

CLAIM 10.2. *Algorithm 4 has time complexity $O(N)$.*

Proof. First, we use Algorithm 2 for the computation of buckets and maxgap values for easy nodes, taking $O(N)$ time (line 1). The total length of the lists $L1$ is $O(N)$, so updating the buckets takes $O(N)$ total time. Hence, it suffices to show that the total time of computing maxgaps is $O(N)$. For $O(d)$ occurrences close to the beginning or the end of γ, we may need to recompute the maxgap from the buckets, each time using $O(N/d)$ time (line 14). Finally, the update in line 16 is performed $O(N)$ times in total and each such step takes $O(1)$ time (note that it suffices to consider only the peripheral elements of each of the buckets).

Thus we obtain the following lemma.

LEMMA 10.2. *For the tree $T(\gamma)$ and a positive integer d, the $[d, 2d]$-restricted quasigaps for all nodes of $T(\gamma)$ can be computed in $O(N)$ time.*

Proof. We compute the $[d, 2d]$-restricted maxgap values for the active nodes of $T(\gamma)$ using Algorithm 4. Given the maxgap values we can compute $[d, 2d]$-restricted quasigap values using formula (6.2) (in $O(1)$ time for each node). For non-active nodes we set $[d, 2d]$-restricted quasigap to "$> 2d$". Claims 10.1 and 10.2 imply that the running time is linear.

11 From quasigaps to seeds.

In this section we show how to reduce finding all seeds of w to computing quasigaps. Let us first describe the way how the set of $O(n^2)$ seeds is represented in $O(n)$ space in the output of the algorithm.

11.1 An $O(n)$ size representation of the set of all seeds

Recall that a similar problem with quasiseeds has a very simple solution: all quasiseeds of w lying on the same edge of the suffix tree form a range with the lower explicit end of the edge being a lower end of the range.

For brevity, by a range we mean here a set of subwords of w lying on the same edge of the suffix tree (that is equivalent) and whose lengths form a range (that is, if v and its prefix u belong to that set then any prefix of v longer than u also does).

Example. For $w = ababaabaab$ (see Fig. 9 for its suffix tree) the sets $\{b\}$, $\{aa, aab\}$ and $\{baba, babaa, babaab\}$ are ranges. The first two have their lower ends explicit, the third does not. On the other hand $\{b, ba\}$ and $\{abab, ababaa\}$ are not ranges.

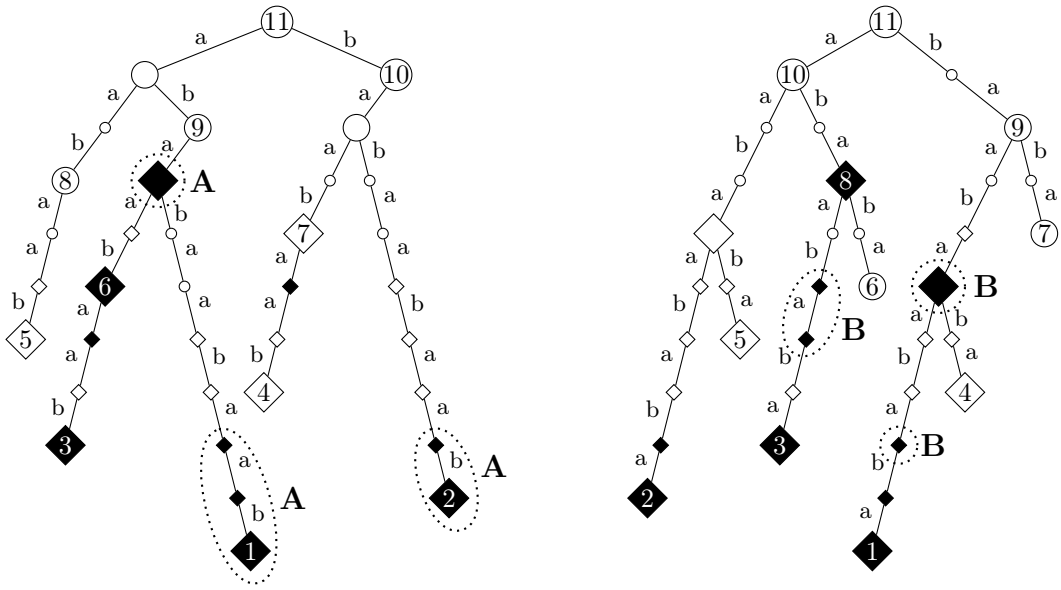

Figure 9: An illustration of quasiseeds, seeds and the output of the algorithm on the suffix trees of w (left) and w^R (right) for $w = ababaabaab$. The explicit nodes are bigger than implicit. The end-marker is omitted; suffixes are represented by the explicit nodes corresponding to them as to ordinary subwords. A node representing the i-th suffix is labelled with i. The quasiseeds are marked as diamonds. The seeds are black and their range representation is marked by dotted ovals. Note that the sets of all seeds on both suffix trees are equal (up to reversing).

With seeds, the problem is much harder than with quasiseeds. They may not form a single range on an edge of the suffix tree, even if we allow both ends to be arbitrary. However, the subwords of w can be represented on the suffix tree of w^R (the reverse of w) as well as of w.

LEMMA 11.1. ([20]) *The set of all seeds can be split into two disjoint classes:* type-A *seeds and* type-B *seeds. Type-A seeds form a single (possibly empty) range on each edge of the suffix tree of w, while type-B seeds on each edge of the suffix tree of w^R. We call such a representation a* range representation *of seeds.*

Note that to use a definition of a range on the suffix tree of w^R for subwords of w one needs to "reverse" all terms.

Example. The word $w = ababaabaab$ (see Fig. 9) has 10 seeds in total. Their range representation is as follows. There are 6 type-A seeds in 3 ranges: $\{aba\}$, $\{ababaaba, ababaabaa, ababaabaab\}$ and $\{babaabaa, babaabaab\}$) and 4 type-B seeds in 3 ranges: $\{baaba, abaaba\}$, $\{abaab\}$ and $\{abaabaab\}$. Note that on the edge $(abaab, abaabaab)$ of the suffix tree of w the set of all seeds is $\{abaaba, abaabaab\}$, which is not a range.

This representation is a useful one as most statistics can be gathered by just iterating through the list of all such non-empty ranges or traversing the suffix trees, obviously both in $O(n)$ time. Finding a shortest seed or all shortest seeds, or counting the number of seeds are just the easiest examples.

11.2 Reduction to the methods of [20] or [7]

Recall that due to Observation 2.1, we need to identify all border seeds among quasiseeds of w. By Observation 2.2, the set of quasiseeds consists of a family of ranges $\{v[1\,..\,k] : \mathsf{quasigap}(v) \le k \le |v|\}$ on an edge (u, v) of T. Such ranges are called *candidate sets* in [20][1].

Now the approach presented in [20] (section "Finding hard seeds"). The following fact is implicitly shown there:

LEMMA 11.2. ([20]) *Assume we have computed the candidate sets of both w and w^R. Then a range representation of all seeds of w can be found in $O(n)$ time.*

Note that we need to run the MAIN procedure twice beforehand to find the quasigaps of explicit nodes of both suffix trees of w and w^R.

[1] Actually, the restrictions on quasiseeds are stronger than ones on candidates in [20]. Hence the candidate sets described above may be smaller than those processed in [20]. However, this does not influence the algorithm as all seeds are still in those sets.

Alternatively, one can exploit a linear time algorithm presented in [7], which, given a family of candidate sets of w, finds a *shortest* border seed of w among those contained in this family, hence a shortest seed.

11.3 Conclusion We have reduced the problem of finding seeds to computing quasigaps of the explicit nodes. Since this can be done in $O(n)$ time (Theorem 7.1), we obtain the main result of the paper.

THEOREM 11.1. *An $O(n)$-size representation of all the seeds of w can be found in $O(n)$ time. In particular, a shortest seed can be computed within the same time complexity.*

12 Proof of Lemma 9.1

Lemma 9.1 is a deep consequence of the definitions of quasigaps, f-factorization, interval staircase and its reduction with respect to a factorization. Its proof is rather intricate, hence we conduct it in several steps.

We start with two simple auxiliary claims. The first one concerns monotonicity of quasigap with respect to the interval. The other binds the quasigaps in two intervals of equal length such that extensions of the corresponding subwords are equal.

CLAIM 12.1. *Let v be a subword of w and $\lambda \subseteq \gamma$ be two intervals. If $Occ(v,\lambda) \neq \emptyset$, then $\mathsf{quasigap}(v,\lambda) \leq \mathsf{quasigap}(v,\gamma)$.*

Proof. We prove the lemma straight from the definition of quasigaps, i.e., the formula (6.2).

Note that $T(\lambda)$ is an induced subtree of $T(\gamma)$, so the explicit nodes of $T(\gamma)$ form a superset of the explicit nodes of $T(\lambda)$. Hence, $|parent(v,\gamma)| \geq |parent(v,\lambda)|$.

Let $\lambda = [i'\,..\,j']$ and $\gamma = [i\,..\,j]$. Observe that the set $Occ(v,\gamma)$ is obtained from $Occ(v,\lambda)$ by adding several elements smaller than i' and greater than j'. Note that in this process the maxgap cannot decrease, therefore $\mathsf{maxgap}(v,\lambda) \leq \mathsf{maxgap}(v,\gamma)$.

Now we show that

$$first(v,\lambda) - i' + 1 \leq \max(\mathsf{maxgap}(v,\gamma),\ first(v,\gamma) - i + 1).$$

Note that $first(v,\lambda) < \infty$, since $Occ(v,\lambda) \neq \emptyset$. Consider two cases. If $first(v,\lambda) = first(v,\gamma)$ then

$$first(v,\lambda) - i' + 1 \leq first(v,\gamma) - i + 1.$$

Otherwise, let $x \in Occ(v,\gamma)$ be the largest element of this set less than i'. Then

$$first(v,\lambda) - i' + 1 \ \leq\ first(v,\lambda) - x \ \leq\ \mathsf{maxgap}(v,\gamma).$$

Similarly, one can prove that

$$\left\lceil \frac{j' - last(v,\lambda)}{2} \right\rceil + 1 \leq \max\left(\mathsf{maxgap}(v,\gamma),\ \left\lceil \frac{j - last(v,\gamma)}{2} \right\rceil + 1\right).$$

CLAIM 12.2. *Let v be a subword of w. Let $0 \leq k \leq n - |v|$ and $1 \leq i, i' \leq n - |v| - k$ be such integers, that*

$$w[i\,..\,i+k+|v|] = w[i'\,..\,i'+k+|v|].$$

Then $\mathsf{quasigap}(v,[i\,..\,i+k]) = \mathsf{quasigap}(v,[i'\,..\,i'+k])$.

Proof. Note that the induced suffix trees $T([i\,..\,i+k])$ and $T([i'\,..\,i'+k])$ are compacted TRIEs of words

$$\{w[i\,..\,n]\#, w[i+1\,..\,n]\#, \ldots, w[i+k\,..\,n]\#\}$$

and

$$\{w[i'\,..\,n]\#, w[i'+1\,..\,n]\#, \ldots, w[i'+k\,..\,n]\#\}$$

respectively. Since $w[i\,..\,i+k+|v|] = w[i'\,..\,i'+k+|v|]$, the top $|v|+1$ levels of $T([i\,..\,i+k])$ and $T([i'\,..\,i'+k])$ are identical. Moreover

$$i+x \in Occ(v,[i\,..\,i+k]) \iff i'+x \in Occ(v,[i'\,..\,i'+k]).$$

Hence, $\mathsf{quasigap}(v,[i\,..\,i+k]) = \mathsf{quasigap}(v,[i'\,..\,i'+k])$, as we were supposed to show.

For the rest of this section let us fix an interval γ with an f-factorization F. Recall that in Section 9 assumed that $m > 0$. What is more we fixed $m = \left\lfloor \frac{n}{50(g+1)} \right\rfloor$. Nevertheless the following claims still hold for any positive integer $0 < m \leq \frac{1}{3}|\gamma|$.

Let $\mathcal{S}$ be an m-staircase covering γ and $\mathcal{K} = Reduce(\mathcal{S}, F)$.

Lemma 9.1, which is the final aim of this section, concerns the reduced staircase. Now we prove a similar result involving the regular m-staircase. It characterizes *all* small quasigaps in γ in terms of quasigaps in an interval staircase covering γ.

CLAIM 12.3. *Let v be a subword of w and let*

$$M = \max\{\mathsf{quasigap}(v,\lambda) : \lambda \in \mathcal{S}\}.$$

If $M \leq m$ then $\mathsf{quasigap}(v,\gamma) = M$, otherwise $\mathsf{quasigap}(v,\gamma) > m$.

First, let us prove the following claim.

CLAIM 12.4. *Let $[i\,..\,j]$ and $[i'\,..\,j']$ be such intervals that $i \leq i' \leq j \leq j'$, $j - i' \geq 2m - 1$, and let*

$$M = \max(\mathsf{quasigap}(v,[i\,..\,j]),\ \mathsf{quasigap}(v,[i'\,..\,j'])).$$

If $M \leq m$, then $\mathsf{quasigap}(v,[i\,..\,j']) = M$, otherwise $\mathsf{quasigap}(v,[i\,..\,j']) > m$.

Proof (of Claim 12.4). Let $O = Occ(v, [i\,..\,j])$, $O' = Occ(v, [i'\,..\,j'])$ and $O'' = Occ(v, [i\,..\,j'])$.

First, assume $\mathsf{quasigap}(v, [i\,..\,j']) \le m$. Then

$$\min(O'') \le i+m-1 \le j \text{ and } \max(O'') \ge j'-2m+2 \ge i'.$$

Hence $O \neq \emptyset$ and $O' \neq \emptyset$. Thus, by Claim 12.1, $M \le \mathsf{quasigap}(v, [i\,..\,j']) \le m$. Therefore if $M > m$ then $\mathsf{quasigap}(v, [i\,..\,j']) \ge m$.

Now, we may assume that $M \le m$. Then, in particular, $O \neq \emptyset$ and $O' \neq \emptyset$. Again by Claim 12.1, $M \le \mathsf{quasigap}(v, [i\,..\,j'])$

Now, it is enough to show that $\mathsf{quasigap}(v, [i\,..\,j']) \le m$ (still assuming $M \le m$). Note that

$$O \cap [i'\,..\,j] = O' \cap [i'\,..\,j] = O \cap O' \neq \emptyset,$$

where the non-emptiness follows from the inequality

$$\min(O') \le i' + m - 1 < j.$$

Therefore

$$\mathsf{maxgap}(O'') = \max(\mathsf{maxgap}(O), \mathsf{maxgap}(O')).$$

Obviously

$$\min(O'') = \min(O) \quad \text{and} \quad \max(O'') = \max(O').$$

Finally, observe that $parent(v, [i\,..\,j'])$ is the lower of the nodes $parent(v, [i\,..\,j])$, $parent(v, [i'\,..\,j'])$ as v is an explicit or implicit node of both induces suffix trees. Hence, by (6.2), $M \le \mathsf{quasigap}(v, [i\,..\,j'])$. This concludes the proof.

Proof (of Claim 12.3). The claim is proved by simple induction over $|\mathcal{S}|$. Since the overlap between each two consecutive intervals in $\mathcal{S}$ is $2m$, the induction step follows from Claim 12.4.

Now, we provide a similar result concerning the reduced staircase instead of the regular staircase. Unfortunately, due to the statement of Claim 12.2 binding the length of v and the length of the subwords of w which we require to be equal, we need to make an additional assumption limiting $|v|$.

Claim 12.5. *Let v be a subword of w such that $|v| \le m$. Additionally, let*

$$M = \max\{\mathit{quasigap}(v, \lambda) : \lambda \in \mathcal{K}\}.$$

If $M \le m$, then $\mathit{quasigap}(v, \gamma) = M$, otherwise $\mathit{quasigap}(v, \gamma) > m$.

Proof. By Claim 12.3, if $\mathsf{quasigap}(v, \gamma) \le m$, then $\mathsf{quasigap}(v, \lambda) \le m$ for all intervals $\lambda \in \mathcal{K}$, consequently $M \le m$. Thus if $M > m$ then $\mathsf{quasigap}(v, \gamma) > m$.

Now, let us assume that $M \le m$. Let the intervals in $\mathcal{S} = \{\beta_1, \ldots, \beta_r\}$ be ordered from left to right, and let us denote $M_p = \max_{h=1,..,p} \mathsf{quasigap}(v, \beta_h)$. From Claim 12.3 we know that $\mathsf{quasigap}(v, \gamma) = M_r$. To prove the lemma, it suffices to show that $M_r = M$.

We show inductively (over p) that

$$M_p = \max\{\mathsf{quasigap}(v, \beta_h) : h \le p \wedge \beta_h \in \mathcal{K}\}.$$

If the last interval β_p belongs to $\mathcal{K}$, the inductive step is clear. Hence, let $\beta_p = [i\,..\,j] \in \mathcal{S} \setminus \mathcal{K}$. By the definition of $\mathcal{K}$, $w[i\,..\,j+m]$ occurs in some factor f_q of F, and the subword f_q occurs in $w[\beta_1 \cup \ldots \cup \beta_{h-1}]$. Hence, $w[i\,..\,j+m] = w[i'\,..\,i'+j-i+m]$, for some $1 \le i' < 2i - j - m$.

By Claim 12.3

$$M_{p-1} = \mathsf{quasigap}(v, \beta_1 \cup \ldots \cup \beta_{p-1}).$$

What is more, $Occ(v, [i'\,..\,i'+j-i]) \neq \emptyset$ since this value is less than m. Thus, by Claims 12.2 and 12.1,

$$\mathsf{quasigap}(v, \beta_p) = \mathsf{quasigap}(v, [i'\,..\,i'+j-i]) \le M_{p-1},$$

and $M_p = M_{p-1}$. Therefore the inductive step is proved. Finally, the induction proves shows $M_r = M$.

The claim above does not provide any information on quasigaps of words longer than m. Such explicit nodes may have implicit node not longer than m in their equivalence class. Hence, this gap needs to be filled, therefore we prove that the assumption limiting the length can be replaced by a similar one involving the length of the parent. This is a major improvement since the length of the parent is a part of the definition of quasigaps. As a consequence we can finally characterize *all* small quasigaps in a succinct way.

Claim 12.6. *Let v be a subword of w, and let*

$$M = \max\{\mathit{quasigap}(v, \lambda) : \lambda \in \mathcal{K}\}.$$

If $|parent(v, \gamma)| < m$ and $M \le m$, then $\mathit{quasigap}(v, \gamma) = M$, otherwise $\mathit{quasigap}(v, \gamma) > m$.

Proof. If $|v| \le m$, also $|parent(v, \gamma)| < m$ and the claim is an obvious consequence of Claim 12.5. So, let us assume that $|v| > m$. If $|parent(v, \gamma)| \ge m$, then, by (6.2), $\mathsf{quasigap}(v, \gamma) > m$. Let us assume that $|parent(v, \gamma)| < m < |v|$, and let $v' = v[1\,..\,m]$. Note that for γ and hence for each $\lambda \in \mathcal{K}$ we have $Occ(v, \lambda) = Occ(v', \lambda)$.

If $M \le m$, then, by the definition of quasigaps, we have $\mathsf{quasigap}(v', \lambda) = \mathsf{quasigap}(v, \lambda)$ for each $\lambda \in \mathcal{K}$. By Claim 12.5, $M = \mathsf{quasigap}(v', \gamma)$, and due to the formula (6.3), $M = \mathsf{quasigap}(v, \gamma)$.

On the other hand, if $M > m$, then $\mathsf{quasigap}(v', \lambda) = \infty$ for some $\lambda \in \mathcal{K}$. From this we conclude that $\mathsf{quasigap}(v', \gamma) = \infty$ and thus $\mathsf{quasigap}(v, \gamma) > |v'| = m$.

Both Claim 12.6 and Lemma 9.1 provide a characterization of all small quasigaps. The first one is probably simpler and cleaner but uses the quasigaps of possibly implicit nodes of $T(\lambda)$ and thus it is the latter that enables *computing* small quasigaps in an *efficient* and fairly simple manner.

LEMMA 9.1. *Assume $m > 0$. Let v be an explicit node in $T(\gamma)$.*
(a) If v is neither an explicit nor an implicit node in $T(\lambda)$ for some $\lambda \in \mathcal{K}$ then $\mathsf{quasigap}(v, \gamma) > m$.
(b) If $|parent(v, \gamma)| \geq m$ then $\mathsf{quasigap}(v, \gamma) > m$.
(c) If the conditions from (a) and (b) do not hold, let

$$M' = \max\{\mathsf{quasigap}(desc(v, \lambda), \lambda) : \lambda \in \mathcal{K}\}.$$

If $M' \leq \min(m, |v|)$ then $\mathsf{quasigap}(v, \gamma) = M'$, otherwise $\mathsf{quasigap}(v, \gamma) > m$.

Proof. Let $M = \max\{\mathsf{quasigap}(v, \lambda) : \lambda \in \mathcal{K}\}$. Let $u = desc(v, \lambda)$. If v is neither an explicit nor an implicit node in $T(\lambda)$ for some $\lambda \in \mathcal{K}$ then $\mathsf{quasigap}(v, \lambda) = \infty$ an thus $M = \infty$. Hence (by Claim 12.6) $\mathsf{quasigap}(v, \gamma) > m$, which concludes part (a) of the lemma. Part (b) is clear from (6.2). Therefore only part (c) remains.

Recall the formula (6.3):

$$\mathsf{quasigap}(v, \gamma) = \begin{cases} \mathsf{quasigap}(u, \gamma) & \text{if } |v| \geq \mathsf{quasigap}(u, \gamma), \\ \infty & \text{otherwise.} \end{cases}$$

Let us assume otherwise, that $v \in T(\lambda)$ for all $\lambda \in \mathcal{K}$. Then we have:

$$M = \begin{cases} M' & \text{if } |v| \geq M', \\ \infty & \text{otherwise.} \end{cases}$$

Indeed, if $M' > |v|$ then, by (6.3), $M = \infty$ and, by Claim 12.6, $\mathsf{quasigap}(v, \gamma) > m$. Otherwise, if $M' \leq |v|$, due to (6.3) we have $M = M'$. Then the statement of the lemma becomes the same as the statement of Claim 12.6.

References

[1] Alberto Apostolico and Andrzej Ehrenfeucht. Efficient detection of quasiperiodicities in strings. *Theor. Comput. Sci.*, 119(2):247–265, 1993.

[2] Alberto Apostolico, Martin Farach, and Costas S. Iliopoulos. Optimal superprimitivity testing for strings. *Inf. Process. Lett.*, 39(1):17–20, 1991.

[3] Omer Berkman, Costas S. Iliopoulos, and Kunsoo Park. The subtree max gap problem with application to parallel string covering. *Inf. Comput.*, 123(1):127–137, 1995.

[4] Dany Breslauer. An on-line string superprimitivity test. *Inf. Process. Lett.*, 44(6):345–347, 1992.

[5] Gerth Stølting Brodal and Christian N. S. Pedersen. Finding maximal quasiperiodicities in strings. In Raffaele Giancarlo and David Sankoff, editors, *CPM*, volume 1848 of *Lecture Notes in Computer Science*, pages 397–411. Springer, 2000.

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

[7] Michalis Christou, Maxime Crochemore, Costas S. Iliopoulos, Marcin Kubica, Solon P. Pissis, Jakub Radoszewski, Wojciech Rytter, Bartosz Szreder, and Tomasz Waleń. Efficient seeds computation revisited. In Raffaelle Giancarlo and Giovanni Manzini, editors, *CPM*, volume 6661 of *Lecture Notes in Computer Science*, pages 350–363. Springer, 2011.

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

[9] Maxime Crochemore. Transducers and repetitions. *Theor. Comput. Sci.*, 45(1):63–86, 1986.

[10] Maxime Crochemore, Christophe Hancart, and Thierry Lecroq. *Algorithms on Strings*. Cambridge University Press, 2007.

[11] Maxime Crochemore, Costas S. Iliopoulos, Marcin Kubica, Wojciech Rytter, and Tomasz Waleń. Efficient algorithms for three variants of the LPF table. *J. Discrete Algorithms*, In Press, Corrected Proof, 2011.

[12] Maxime Crochemore and Wojciech Rytter. *Jewels of Stringology*. World Scientific, 2003.

[13] Maxime Crochemore and German Tischler. Computing longest previous non-overlapping factors. *Inf. Process. Lett.*, 111(6):291–295, 2011.

[14] Martin Farach. Optimal suffix tree construction with large alphabets. In *FOCS*, pages 137–143, 1997.

[15] H. N. Gabow and R. E. Tarjan. A linear-time algorithm for a special case of disjoint set union. *Proceedings of the 15th Annual ACM Symposium on Theory of Computing (STOC)*, pages 246–251, 1983.

[16] Qing Guo, Hui Zhang, and Costas S. Iliopoulos. Computing the *ambda*-seeds of a string. In Siu-Wing Cheng and Chung Keung Poon, editors, *AAIM*, volume 4041 of *Lecture Notes in Computer Science*, pages 303–313. Springer, 2006.

[17] Qing Guo, Hui Zhang, and Costas S. Iliopoulos. Computing the lambda-covers of a string. *Inf. Sci.*, 177(19):3957–3967, 2007.

[18] Dov Harel and Robert Endre Tarjan. Fast algorithms for finding nearest common ancestors. *SIAM J. Comput.*, 13(2):338–355, 1984.

[19] Costas S. Iliopoulos, Manal Mohamed, and William F.

Smyth. New complexity results for the k-covers problem. *Inf. Sci.*, 181(12):2571–2575, 2011.

[20] Costas S. Iliopoulos, D. W. G. Moore, and Kunsoo Park. Covering a string. *Algorithmica*, 16(3):288–297, 1996.

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

[22] Costas S. Iliopoulos and William F. Smyth. An on-line algorithm of computing a minimum set of k-covers of a string. In *Proc. of the Ninth Australian Workshop on Combinatorial Algorithms (AWOCA)*, pages 97–106, 1998.

[23] Roman M. Kolpakov and Gregory Kucherov. Finding maximal repetitions in a word in linear time. In *Proceedings of the 40th Symposium on Foundations of Computer Science*, pages 596–604, 1999.

[24] Roman M. Kolpakov and Gregory Kucherov. Finding maximal repetitions in a word in linear time. In *FOCS*, pages 596–604, 1999.

[25] Yin Li and William F. Smyth. Computing the cover array in linear time. *Algorithmica*, 32(1):95–106, 2002.

[26] M. Lothaire, editor. *Algebraic Combinatorics on Words*. Cambridge University Press, 2001.

[27] Michael G. Main. Detecting leftmost maximal periodicities. *Discrete Applied Mathematics*, 25(1-2):145–153, 1989.

[28] Dennis Moore and William F. Smyth. Computing the covers of a string in linear time. In *SODA*, pages 511–515, 1994.

[29] 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.

[30] W. F. Smyth. Repetitive perhaps, but certainly not boring. *Theor. Comput. Sci.*, 249(2):343–355, 2000.

[31] Jacob Ziv and Abraham Lempel. A universal algorithm for sequential data compression. *IEEE Transactions on Information Theory*, 23(3):337–343, 1977.