



HAL
open science

Quasi-Linear-Time Algorithm for Longest Common Circular Factor

Mai Alzamel, Maxime Crochemore, Costas S Iliopoulos, Tomasz Kociumaka, Jakub Radoszewski, Wojciech Rytter, Juliusz Straszyski, Tomasz Waleń, Wiktor Zuba

► **To cite this version:**

Mai Alzamel, Maxime Crochemore, Costas S Iliopoulos, Tomasz Kociumaka, Jakub Radoszewski, et al.. Quasi-Linear-Time Algorithm for Longest Common Circular Factor. 30th Annual Symposium on Combinatorial Pattern Matching (CPM 2019), Nadia Pisanti and Solon P. Pissis, Jun 2019, Pisa, Italy. pp.25:1–25:14, 10.4230/LIPIcs.CPM.2019.25 . hal-04338883

HAL Id: hal-04338883

<https://hal.science/hal-04338883>

Submitted on 12 Dec 2023

HAL is a multi-disciplinary open access archive for the deposit and dissemination of scientific research documents, whether they are published or not. The documents may come from teaching and research institutions in France or abroad, or from public or private research centers.


L'archive ouverte pluridisciplinaire **HAL**, est destinée au dépôt et à la diffusion de documents scientifiques de niveau recherche, publiés ou non, émanant des établissements d'enseignement et de recherche français ou étrangers, des laboratoires publics ou privés.

1 Quasi-Linear-Time Algorithm for Longest 2 Common Circular Factor

3 **Mai Alzamel**

4 Department of Informatics, King's College London, London, UK

5 mai.alzamel@kcl.ac.uk

6  <https://orcid.org/0000-0002-7590-9919>

7 **Maxime Crochemore**


8 Department of Informatics, King's College London, London, UK

9 maxime.crochemore@kcl.ac.uk

10 **Costas S. Iliopoulos**

11 Department of Informatics, King's College London, London, UK


12 costas.ilopoulos@kcl.ac.uk

13  <https://orcid.org/0000-0003-3909-0077>

14 **Tomasz Kociumaka**

15 Institute of Informatics, University of Warsaw, Warsaw, Poland

16 kociumaka@mimuw.edu.pl

17  <https://orcid.org/0000-0002-2477-1702>

18 **Jakub Radoszewski**

19 Institute of Informatics, University of Warsaw, Warsaw, Poland

20 jrad@mimuw.edu.pl

21  <https://orcid.org/0000-0002-0067-6401>

22 **Wojciech Rytter**

23 Institute of Informatics, University of Warsaw, Warsaw, Poland

24 rytter@mimuw.edu.pl

25  <https://orcid.org/0000-0002-9162-6724>

26 **Juliusz Straszynski**

27 Institute of Informatics, University of Warsaw, Warsaw, Poland


28 j.straszynski@mimuw.edu.pl

29  <https://orcid.org/0000-0003-2207-0053>

30 **Tomasz Waleń**

31 Institute of Informatics, University of Warsaw, Warsaw, Poland

32 walen@mimuw.edu.pl

33  <https://orcid.org/0000-0002-7369-3309>

34 **Wiktor Zuba**

35 Institute of Informatics, University of Warsaw, Warsaw, Poland

36 w.zuba@mimuw.edu.pl

37 — Abstract —

38 We introduce the Longest Common Circular Factor (LCCF) problem in which, given strings S
39 and T of length n , we are to compute the longest factor UV of string S whose cyclic shift VU
40 occurs as a factor of string T . The LCCF is a new similarity measure that is an extension of the
41 classic Longest Common Factor problem to circular pattern matching. We show that the LCCF
42 problem can be solved in $\tilde{O}(n)$ time¹. Our algorithm is randomized (Las Vegas). We apply

¹ The \tilde{O} notation ignores factors polylogarithmic with respect to the instance size (encoded in binary).

43 techniques of internal pattern matching (in case U and V are not highly periodic) and Lyndon
 44 roots (otherwise) and reduce the problem to computing an intersection of 4D rectangles.

45 **2012 ACM Subject Classification** Theory of computation → Pattern matching

46 **Keywords and phrases** longest common factor, circular pattern matching, internal pattern
 47 matching, intersection of hyperrectangles

48 1 Introduction

49 We introduce a new variant of the Longest Common Factor (LCF) Problem, called the
 50 Longest Common Circular Factor (LCCF) Problem. In the LCCF problem, given two strings
 51 S and T , both of length n , we seek for the longest factor of S whose cyclic shift occurs as a
 52 factor of T . The LCCF is a new string similarity measure that is 2-approximated by the
 53 LCF. We show that the exact value of LCCF be computed efficiently.

54 A linear-time solution to the LCF problem is one of the best known applications of the
 55 suffix tree [2]. Just as the LCF problem was an extension of the classical pattern matching,
 56 the LCCF can be seen as an extension of the circular pattern matching problem. The latter
 57 can also be solved in linear time using the suffix tree and also admits a number of efficient
 58 solutions based on practical approaches [4, 9, 10, 11, 16, 20, 24, 27], also in the approximate
 59 variant [6, 7, 17, 19], as well as an indexing variants [3, 20, 21] and the problem of detecting
 60 various circular patterns [25]. The LCCF problem is also related to the notion of unbalanced
 61 translocations [8, 12, 26, 28, 29].

62 The problem in scope can be formally stated as follows.

LONGEST COMMON CIRCULAR FACTOR (LCCF)

Input: Two strings S and T of length n each

Output: A longest pair of factors, F of S and F' of T , for which there exist strings U
 and V such that $F = UV$ and $F' = VU$; we denote $\text{LCCF}(S, T) = (F, F')$

64 Our main result is the following.

65 **Main result.** The LCCF problem can be solved in $\mathcal{O}(n \log^6 n)$ time by a randomized (Las
 66 Vegas) algorithm.

67 For simplicity, we will only consider computing the length of the LCCF. However, a
 68 corresponding pair of factors can be retrieved in a straightforward way from the algorithm.

69 2 Preliminaries

70 We consider strings over an integer alphabet Σ . If W is a string, then by $|W|$ we denote
 71 its length and by $W[1], \dots, W[|W|]$ its characters. By $x = W[i..j]$ we denote a *fragment*
 72 of W between the i th and j th character, inclusively. If $i = 1$, it is called a prefix, and if
 73 $j = |W|$, it is called a suffix. The string $W[i] \dots W[j]$ that corresponds to the fragment x is
 74 called a *factor* of W . We say that two fragments *match* if their corresponding factors are the
 75 same. By $W[i..j]$ we denote the fragment $W[i..j-1]$. Fragments are further denoted by
 76 lowercase letters. Let us note that a factor of a given string that is specified by a fragment
 77 can be represented by its endpoints in $\mathcal{O}(1)$ space.

78 By W^R we denote the reversal of the string W . By $\text{per}(W)$ we denote the shortest period
 79 of W . String W is *highly periodic* iff $\text{per}(W) \leq |W|/3$.

80 By the *type* of a fragment u ($\text{type}(u)$) we mean the largest k such that $2^k \leq |u|$. We denote
 81 by $\text{LCCF}_{a,b}(S, T) = (F, F')$ the longest common circular factor of S, T such that $F = UV$,
 82 $F' = VU$, U is of type a , and V is of type b . We also say that it is the type- (a, b) LCCF.
 83 Our strategy is to compute $\text{LCCF}_{a,b}(S, T) = (F, F')$ independently for each $a, b \leq \log n$, and
 84 afterwards compute the longest alternative (over all pairs (a, b)) as the final result.

85 2.1 Synchronizing Functions

86 Let W be a string of length n . By $\mathcal{F}_t(W)$ we denote the set of fragments of W of length
 87 t and by $\mathcal{N}_t(W)$ we denote the set of non-highly-periodic fragments of W of length t . By
 88 $\mathcal{F}_t(x)$ and $\mathcal{N}_t(x)$ we denote subsets of these sets defined for a fragment x of W .

89 A function $\text{sync} : \mathcal{F}_{2\tau-1}(W) \mapsto \mathcal{F}_\tau(W) \cup \{\perp\}$ is called τ -synchronizing if it satisfies the
 90 following conditions for each fragment $x \in \mathcal{F}_{2\tau-1}(W)$ (see [22]):

- 91 ■ If $\text{sync}(x) = \perp$, then $\mathcal{N}_\tau(x) = \emptyset$;
- 92 ■ If $\text{sync}(x) \neq \perp$, then $\text{sync}(x) \in \mathcal{N}_\tau(x)$;
- 93 ■ If two fragments $x, x' \in \mathcal{F}_{2\tau-1}(W)$ are matching and $\text{sync}(x) = x[s..s+\tau]$ then $\text{sync}(x') =$
 94 $x'[s..s+\tau]$ for the same s . In other words, if $x = W[p..q]$ and $x' = W[p'..q']$, then
 95 $\text{sync}(x) = W[p+s..p+s+\tau]$ and $\text{sync}(x') = W[p'+s..p'+s+\tau]$.

96 The elements of $\mathcal{F}_\tau(W)$ for $\tau = 2^k$ are called here τ -basic fragments.

97 ► **Example 1.** Let $\text{DBF}(u)$ be the identifier of a τ -basic fragment u of W in the *Dictionary*
 98 *of Basic Factors*, see [14], and π be a permutation of all (linearly many) τ -basic identifiers.
 99 Each identifier is an integer in the range $[1..n]$. For a fragment x of size $2\tau - 1$, we could
 100 define $\text{sync}(x)$ as the first τ -basic fragment u (from the left) with minimmal $\text{DBF}(u)$. Then
 101 sync satisfies the conditions of the synchronizing function. If we take a random permutation
 102 π , then it has other useful properties in expectation, as shown in the lemma below.

103 The set of τ -synchronizers for a fragment x , denoted by $\text{SYNC}_\tau(x)$, is $\text{sync}(\mathcal{F}_{2\tau-1}(W)) \cap$
 104 $\mathcal{N}_\tau(x)$. It follows from the construction of a τ -synchronizing function of [22] and of [23].

105 ► **Lemma 2.** For all $\tau = 2^i$, $\tau \leq n$, one can construct in $\mathcal{O}(n \log n)$ total time a τ -
 106 synchronizing function (stored in an array) such that for each fragment x the expected number
 107 of τ -synchronizers is $\mathcal{O}(|x|/\tau)$.

108 **Proof.** Let $\mathcal{B}_\tau(S) = \{y \in \mathcal{N}_\tau(S) : \text{per}(y[1.. \tau - 1]) \leq \frac{\tau}{3} \text{ or } \text{per}(y[2.. \tau]) \leq \frac{\tau}{3}\}$. A set of
 109 integers A is called d -sparse if $\min\{|a - b| : a, b \in A, a \neq b\} \geq d$. The following claim was
 110 presented as [22, Lemma 4.4.7].

111 ► **Claim 3.** $\mathcal{B}_\tau(S)$ is a union of two $\frac{\tau}{3}$ -sparse sets.

112 For a step function f on integers, by $\text{Steps}(f)$ we denote the number of positions such
 113 that $f(x) \neq f(x + 1)$. The following fact shows the existence of a τ -synchronizing function
 114 with a small number of steps and that it can be constructed efficiently. It was presented as
 115 Lemma 4.4.8 in [22]; it is also present in the non-periodic case of [23].

116 ► **Claim 4.** For a string S of length n , there exists a τ -synchronizing function sync such that
 117 $\text{Steps}(\text{sync}(\mathcal{F}_{2\tau-1}(S))) = \mathcal{O}(n/\tau)$. Moreover, this function satisfies $\mathbb{P}[\text{sync}(x) = y] \leq \frac{3}{\tau}$ for
 118 $y \in \mathcal{N}_\tau(x) \setminus \mathcal{B}_\tau(x)$ and it can be constructed (as an array) in $\mathcal{O}(n)$ time.

119 We can now proceed to the proof of the lemma. The construction of a τ -synchronizing
 120 function of Claim 4 can be applied. The fact that $|\text{SYNC}_\tau(x) \cap \mathcal{B}_\tau(S)| = \mathcal{O}(|x|/\tau)$ in
 121 expectation follows from Claim 3 and the fact that $|\text{SYNC}_\tau(x) \cap (\mathcal{N}_\tau \setminus \mathcal{B}_\tau(S))| = \mathcal{O}(|x|/\tau)$ in
 122 expectation follows from Claim 4. ◀

123 **3 Nonperiodic-Nonperiodic Case**

124 We say that a string U of type a is a -highly periodic if it has a period that is at most
 125 $2^{a-1}/3$. We consider now $\text{LCCF}_{a,b}(S, T) = (F, F')$ such that $F = UV$, $F' = VU$, U is of
 126 type a , V is of type b , U is not a -highly periodic, and V is not b -highly periodic. We call it
 127 *nonperiodic-nonperiodic* case.

128 We extend the function sync to all fragments by defining $\text{sync}(u)$ as the 2^{k-1} -synchronizer
 129 of the prefix of u of size 2^k , where $k = \text{type}(u)$.

130 Let us define basic fragments called the *left k -window* and the *right k -window*:

131 $\text{LeftWin}_k(W, i) = W[i - 2^{k+1} .. i - 1]$, $\text{RightWin}_k(W, i) = W[i .. i + 2^{k+1} - 1]$

132 For two triples $\Delta_1 = (x, i, y)$, $\Delta_2 = (y, i', x)$, where x is a fragment both in $\text{LeftWin}(i, S)$
 133 and in $\text{RightWin}_a(i', T)$ and y is a fragment both in $\text{RightWin}(i, S)$ and in $\text{LeftWin}_b(i', T)$
 134 define the pair $\text{CAND}(\Delta_1, \Delta_2) = (F, F')$ of fragments in S and T as follows, see Figure 1.

Relation of pairs of triples (Δ_1, Δ_2) to $\text{LCCF}_{a,b}$

Let j, k be starting positions of fragments x, y in S .

Let j', k' be starting positions of fragments x, y in T .

135 $l := j' - i' + i - j$, $r := k - i + i' - k'$

Then $\text{CAND}(\Delta_1, \Delta_2) = (F, F')$, where

$$F := S[i - l .. i + r], F' := T[i' - r .. i' + l].$$

(F, F') is a candidate for $\text{LCCF}_{a,b}(S, T)$

136 Due to synchronization for any pair of a, b -triples $\Delta_1 = (x, i, y)$, $\Delta_2 = (y, i', x)$ there are
 137 exactly two fragments U, V of types a and b , such that UV is "anchored" at i in S and VU is
 138 "anchored" at i' in T , $\text{sync}(U) = x$, $\text{sync}(V) = y$ and $[UV, VU]$ is a potential common circular
 139 factor. If $\text{CAND}(\Delta_1, \Delta_2) = (F, F')$, then $(F, F') = (UV, VU)$, $l = |U|$, $r = |V|$.

140 For a string W we introduce the following sets of 2^{k-1} -synchronizers:

141 $\text{LeftSynch}_k(W, i) = \text{SYNC}_{2^{k-1}}(\text{LeftWin}_{k+1}(W, i))$

142 $\text{RightSynch}_k(W, i) = \text{SYNC}_{2^{k-1}}(\text{RightWin}_{k+1}(W, i))$

143 Define $\text{Triples}_{c,d}(W) = \{(x, i, y) : x \in \text{LeftSynch}_c(W, i), y \in \text{RightSynch}_d(W, i), i \in$
 144 $[1 .. |W|]\}$.

145 Using the terminology of triples and function CAND (giving correspondence between pairs
 146 of triples and pairs of fragments), a scheme of a general algorithm can be written informally
 147 as follows:

Algorithm Compute-LCCF $_{a,b}(S)$

1. Compute the sets $\text{Triples}_{a,b}(S)$ and $\text{Triples}_{b,a}(T)$.
2. Find a pair of triples $\Delta_1 \in \text{Triples}_{a,b}(S)$, $\Delta_2 \in \text{Triples}_{b,a}(T)$
 such that $\text{CAND}(\Delta_1, \Delta_2)$ corresponds to $\text{LCCF}_{a,b}(S, T)$.

149 **► Lemma 5.** *In the nonperiodic-nonperiodic case the algorithm Compute-LCCF $_{a,b}$ correctly*
 150 *computes $\text{LCCF}_{a,b}(S, T)$.*

151 However there are two difficulties in this approach that need to be overcome if the complexity
 152 should be $\mathcal{O}(n)$.

153 Firstly, we cannot consider individually each combination, since the number of possible
 154 pairs (i, i') is quadratic. We overcome this problem by a reduction to a geometric problem of
 155 the intersection of families of 4D rectangles.

156 Secondly, there is a difficulty of dealing with the case of highly periodic U or V . We
 157 overcome it by using the machinery of *runs* and *Lyndon roots* of strings. In periodic case the
 158 role of synchronizers is played by fragments which are Lyndon roots.

159 4 4-Families-Problem

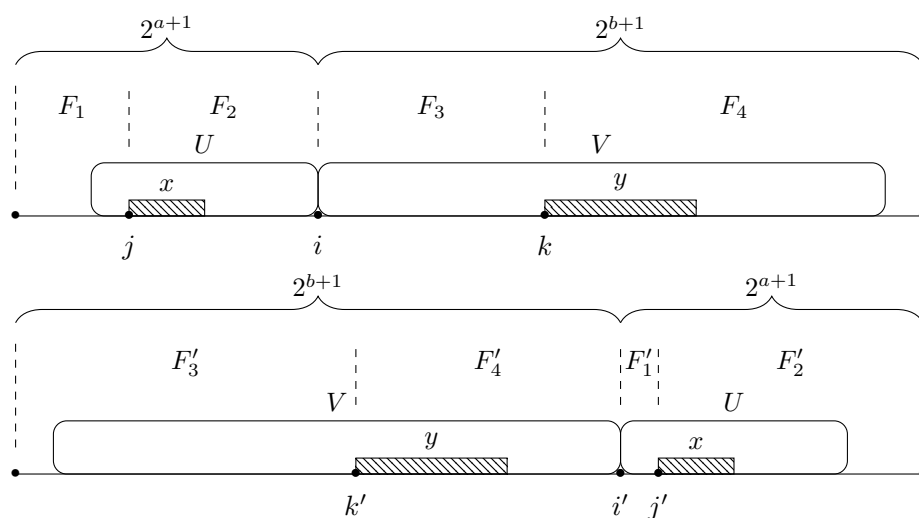
160 We reduce the second statement in the algorithm *Compute-LCCF_{a,b}* to 4-FAMILIES-PROBLEM.
 161 A 4-tuple (F_1, F_2, F_3, F_4) of strings agrees with a 4-tuple (F'_1, F'_2, F'_3, F'_4) iff

162 F'_1 is a suffix of F_1 , F_2 is a prefix of F'_2
 163 F_3 is a suffix of F'_3 , F'_4 is a prefix of F_4 ;

163 see Figure 1.

164 Our main auxiliary problem is defined as follows:

4-FAMILIES-PROBLEM
Input: A family \mathcal{F}_S of 4-tuples (F_1, F_2, F_3, F_4) of factors of S and a family \mathcal{F}_T of
 165 4-tuples (F'_1, F'_2, F'_3, F'_4) of factors of T , where $m = |\mathcal{F}_S| + |\mathcal{F}_T|$
Output: A 4-tuple $(F_1, F_2, F_3, F_4) \in \mathcal{F}_S$ which agrees with $(F'_1, F'_2, F'_3, F'_4) \in \mathcal{F}_T$ that
 maximizes $|F'_1| + |F_2| + |F_3| + |F'_4|$



165 **Figure 1** (F_1, F_2, F_3, F_4) agrees with (F'_1, F'_2, F'_3, F'_4) iff $\text{CAND}((x, i, y), (y, i', x))$ corresponds to a common circular factor UV , where $U = F'_1 F_2$, $V = F_3 F'_4$.

166 Denote $\text{Win}_{a,b}(W, i) = \text{LeftWin}_a(W, i)\text{RightWin}_b(W, i)$.

For each triple $(x, i, y) \in \text{Triples}_{a,b}(W)$ such that x starts at position j and y starts at position k we denote by $\Psi_{a,b}(x, i, y)$ the following factorization (F_1, F_2, F_3, F_4) of $\text{Win}_{a,b}(W, i)$:

$$F_1 = S[i - 2^{a+1} + 1 .. j], \quad F_2 = S[j .. i], \quad F_3 = S[i .. k], \quad F_4 = S[k .. i + 2^{b+1}].$$

167 Moreover, by $\Psi'_{a,b}(x, i, y)$ we denote (F_3, F_4, F_1, F_2) .

168 ► **Observation 6.** For two triples $\Delta_1 = (x, i, y)$, $\Delta_2 = (y, i', x)$ $\text{CAND}(\Delta_1, \Delta_2)$ is a common
 169 circular factor iff $\Psi(\Delta_1)$ agrees with $\Psi'(\Delta_2)$.

170 In the lemma below *with high probability* means, as usually, that the error probability is
 171 $\mathcal{O}(n^{-c})$ for any specified constant $c > 0$.

172 ► **Proposition 7.** After $\mathcal{O}(n \log^2 n)$ -time preprocessing, in the nonperiodic-nonperiodic case
 173 the $\text{LCCF}_{a,b}$ problem can be reduced to a 4-FAMILIES-PROBLEM($\mathcal{F}_S, \mathcal{F}_T$) in which $m =$
 174 $\mathcal{O}(n \log n)$ with high probability.

175 **Proof.** If $(F_1, F_2, F_3, F_4) \in \mathcal{F}_1$ agrees with $(F'_1, F'_2, F'_3, F'_4) \in \mathcal{F}_2$, then S and T contain a
 176 common circular factor of length $|F'_1| + |F_2| + |F_3| + |F'_4|$; see Fig. 1. We will now show that
 177 if $\text{LCCF}_{a,b}(S, T) = (F, F')$, then the solution to 4-FAMILIES-PROBLEM($\mathcal{F}_1, \mathcal{F}_2$) is at least $|F|$.

Assume that $F = x_1 x_2$ and $F' = y_2 y_1$ where x_1 matches y_1 , x_2 matches y_2 , $2^a \leq |x_1| \leq$
 $2^{a+1} - 1$, and $2^b \leq |x_2| \leq 2^{b+1} - 1$. Let i be the starting position of fragment x_2 and i'
 be the starting position of the fragment y_1 . Let $\tau_1 = 2^{a-1}$, $\tau_2 = 2^{b-1}$, x'_1 and y'_1 be the
 prefixes of x_1 and y_1 , respectively, of length $2\tau_1 - 1$, x'_2 and y'_2 be the prefixes of x_2 and y_2 ,
 respectively, of length $2\tau_2 - 1$, and:

$$x = \text{sync}_{\tau_1}(x'_1) = \text{sync}_{\tau_1}(y'_1), \quad y = \text{sync}_{\tau_2}(x'_2) = \text{sync}_{\tau_2}(y'_2).$$

Let j and k' be the starting positions of the synchronizing occurrences of x in x'_1 and y'_1 and
 k and j' be the starting positions of the synchronizing occurrences of y in x'_2 and y'_2 . Then
 in the above construction, we will have (F_1, F_2, F_3, F_4) in \mathcal{F}_1 where

$$F_1 = S[i - 2^{a+1} + 1 .. j], \quad F_2 = S[j .. i], \quad F_3 = S[i .. k], \quad F_4 = S[k .. i + 2^{b+1}).$$

and (F'_1, F'_2, F'_3, F'_4) in \mathcal{F}_2 where

$$F'_3 = T[i' - 2^{a+1} + 1 .. j'], \quad F'_4 = T[j' .. i'], \quad F'_1 = T[i' .. k'], \quad F'_2 = T[k' .. i' + 2^{b+1}).$$

178 Moreover, (F_1, F_2, F_3, F_4) agrees with (F'_1, F'_2, F'_3, F'_4) due to the existence of (F, F') and its
 179 weight $|F'_1| + |F_2| + |F_3| + |F'_4|$ equals $|F| = |F'|$.

180 The preprocessing of Lemma 2 takes $\mathcal{O}(n \log^2 n)$ time. Afterwards, for given a and b , the
 181 above construction produces, with high probability, two families $\mathcal{F}_1, \mathcal{F}_2$ with $|\mathcal{F}_1|, |\mathcal{F}_2| =$
 182 $\mathcal{O}(n)$. ◀

183 5 Periodic-Periodic Case

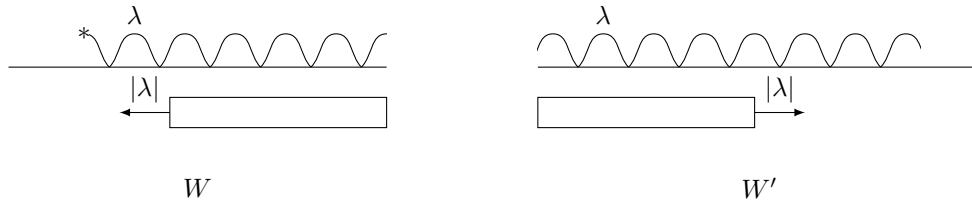
184 We consider now $\text{LCCF}_{a,b}(S, T) = (F, F')$ such that $F = UV$, $F' = VU$, U is of type a , V is
 185 of type b , U is a -highly periodic, and V is b -highly periodic.

186 For a string W , by $\text{HPerPref}(U)$ and $\text{HPerSuf}(W)$ we denote the maximal highly periodic
 187 prefix and suffix of U . Let us start with the following simple observation; see Fig. 2.

188 ► **Observation 8.** Let W and W' be two strings for which the strings $X = \text{HPerSuf}(W)$ and
 189 $Y = \text{HPerPref}(W')$ have the same Lyndon root λ . Then the longest highly periodic suffix of
 190 W that is also a prefix of W' has length at least $\min(|X|, |Y|) - |\lambda|$.

191 Let us recall that Lyndon roots of runs can be computed in constant time after linear-time
 192 preprocessing [13] and that the 2-period query problem can be solved within the same
 193 complexities [23] (for a simplified solution of this problem, see [5]).

For a fragment u denote by $\text{Lyndons}(u)$ the set of fragments in u corresponding to the
 first/second/last occurrence of Lyndon root in the maximal highly periodic suffix of u (if

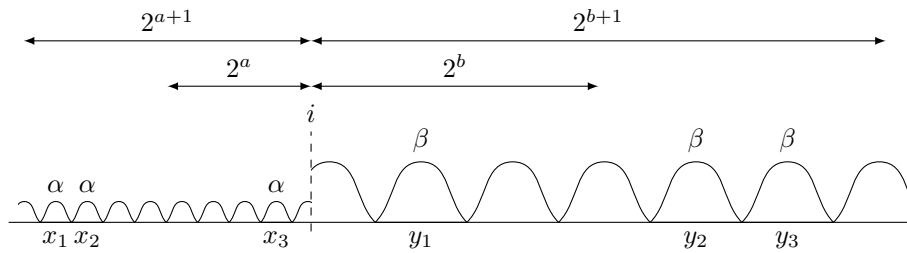


■ **Figure 2** Illustration of Observation 8.

there is any such prefix/suffix) and by $\text{Lyndons}'(u)$ the set of fragments in u corresponding to the first/penultimate/last occurrence of Lyndon root in the maximal highly periodic prefix of u (if there is any such prefix/suffix). We can redefine

$$\text{Triples}_{a,b}(S) = \{ (x, i, y) : x \in \text{Lyndons}(\text{LeftWin}_a(i)), y \in \text{Lyndons}'(\text{RightWin}_b(i)) \};$$

194 see Fig. 3.



■ **Figure 3** $\text{Triples}_{a,b}(S)$ for position i contains (x_p, i, y_q) for $p, q \in \{1, 2, 3\}$.

195 ▶ **Lemma 9.** *The algorithm $\text{Compute-LCCF}_{a,b}(S)$ for redefined triples correctly solves the*
 196 *periodic-periodic case.*

197 Let us redefine: $\mathcal{F}_S = \{ \Psi(x, i, y) \in \text{Triples}_{a,b}(S) \}$, $\mathcal{F}_T = \{ \Psi'(x, i, y) \in \text{Triples}_{a,b}(T) \}$

198 ▶ **Proposition 10.** *In the periodic-periodic case $\text{LCCF}(S, T)$ corresponds to a solution of*
 199 *4-FAMILIES-PROBLEM($\mathcal{F}_S, \mathcal{F}_T$) for $\mathcal{F}_S, \mathcal{F}_T$ defined above.*

200 6 Nonperiodic-Periodic Case

201 Finally we consider the case that $\text{LCCF}_{a,b}(S, T) = (F, F')$ such that $F = UV$, $F' = VU$, U
 202 is of type a , V is of type b , and either U is a -highly periodic or V is b -highly periodic. This
 203 case can be reduced to 4-FAMILIES-PROBLEM directly by combining the techniques of the
 204 previous two sections.

205 ▶ **Proposition 11.** *After $\mathcal{O}(n \log^2 n)$ -time preprocessing, in the nonperiodic-nonperiodic*
 206 *case the $\text{LCCF}_{a,b}$ problem can be reduced to a 4-FAMILIES-PROBLEM($\mathcal{F}_S, \mathcal{F}_T$) in which*
 207 *$m = \mathcal{O}(n \log n)$ with high probability.*

208 **7** Solution to 4-Families-Problem

209 In this section we show how to solve the 4-FAMILIES-PROBLEM.

210 **7.1 Intersecting 4D Rectangles**

211 We define a *d-rectangle* ($d \geq 2$) as a Cartesian product of d closed intervals, such that at
 212 least $d - 2$ of them are singletons. E.g., $\{3\} \times [2, 5] \times [1, 7] \times \{0\}$ is a 4-rectangle. In other
 213 words, a *d-rectangle* is an isothetic hyperrectangle of dimension at most 2. Two *d-rectangles*
 214 $[a_1, b_1] \times \cdots \times [a_d, b_d]$ and $[a'_1, b'_1] \times \cdots \times [a'_d, b'_d]$ are called *compatible* if, for each $i \in \{1, \dots, d\}$,
 215 $[a_i, b_i]$ or $[a'_i, b'_i]$ is a singleton.

216 We consider two families of 4-rectangles with weights and wish to find a pair of intersecting
 217 rectangles, one per family, with maximum total weight. The general problem of finding
 218 such an intersection of two families of m weighted hyperrectangles in d dimensions can be
 219 solved in $\mathcal{O}(m \log^{2d} m)$ time by an adaptation of a classic approach [15]. Below we consider
 220 a special variant of the problem that has a much more efficient solution.

MAX-WEIGHT INTERSECTION OF COMPATIBLE RECTANGLES IN 4D

221 **Input:** Two families \mathcal{R}_1 and \mathcal{R}_2 of 4-rectangles in \mathbb{Z}^4 with integer weights containing
 m rectangles in total, such that each $R_1 \in \mathcal{R}_1$ and $R_2 \in \mathcal{R}_2$ are compatible

Output: Check if there is an intersecting pair of 4-rectangles $R_1 \in \mathcal{R}_1$ and $R_2 \in \mathcal{R}_2$
 and, if so, compute the maximum total weight of such a pair

222 A very similar problem was considered as Problem 3 in [18] for an arbitrary d . The sole
 223 difference is that the weight of an intersection of two *d-rectangles* $R_1 \in \mathcal{R}_1$ and $R_2 \in \mathcal{R}_2$ in
 224 that problem was the maximum ℓ_1 -norm of a point in $R_1 \cap R_2$. A solution to Problem 3
 225 for $d = 4$ in the case that the 4-rectangles are compatible working in $\mathcal{O}(m \log^3 m)$ time and
 226 $\mathcal{O}(m \log^2 m)$ space was given as [18, Lemma 5.8]. The algorithm presented in that lemma
 227 actually solves the MAX-WEIGHT INTERSECTION OF COMPATIBLE RECTANGLES IN 4D
 228 problem and applies it for specific weight assignment of the 4-rectangles on the input. It
 229 uses hyperplane sweep and a variant of an interval stabbing problem. Henceforth we will use
 230 the following result.

231 **► Fact 12** ([18, see Lemma 5.8]). MAX-WEIGHT INTERSECTION OF COMPATIBLE RECT-
 232 ANGLES IN 4D can be solved in $\mathcal{O}(m \log^3 m)$ time and $\mathcal{O}(m \log^2 m)$ space.

233 **7.2 Computations on Factor Intervals**

234 The main tool in the solution are factor intervals. Before we describe them and how we
 235 operate on them, let us recall the basic properties of the suffix tree. Let us start with the
 236 following known fact.

237 **► Fact 13** ([1]). Let W be a string of length n and \mathcal{T} be its suffix tree. After $\mathcal{O}(n)$ -time
 238 preprocessing, given a fragment x of W , one can compute its locus in \mathcal{T} in $\mathcal{O}(\log \log n)$ time.

239 Let W be a string of length n . Let us consider the lexicographic order of factors of W .
 240 For two strings X and Y such that $X \leq Y$, by $[X, Y]$ we denote the set of factors F of W
 241 such that $X \leq F \leq Y$. This set is referred to as a *factor interval*.

242 Similarly as for integers, one can define a *factor hyperrectangle* as a Cartesian product of
 243 factor intervals and a *factor d-rectangle* as a Cartesian product of d factor intervals such that
 244 at most two of them are not singletons. In our algorithm we will construct factor intervals

245 and factor 4-rectangles and use the following lemma to transform them into integer intervals
246 and 4-rectangles.

247 ► **Lemma 14.** *Let W be a string of length n and \mathcal{A} be a family of factor intervals whose
248 endpoints are factors of W . In $\mathcal{O}(n + |\mathcal{A}| \log \log n)$ time one can construct a function f
249 that maps factor intervals into integer intervals such that, for any $P, Q \in \mathcal{A}$, $P \cap Q \neq \emptyset$ iff
250 $f(P) \cap f(Q) \neq \emptyset$. Moreover, if P is a singleton, so is $f(P)$.*

251 **Proof.** First we preprocess string W to support lexicographically minimal factors of W
252 starting with given fragment x using Lemma 16. This step takes $\mathcal{O}(n)$ time.

253 For each factor X of string W , we define function g as $g(X) = (r, |X|)$ where r is a rank
254 of the lexicographically minimal suffix of W starting with fragment of X . We can observe
255 that for any factors X, Y , $X \leq Y$ iff $g(X) \leq g(Y)$.

256 Since we are interested only in values of function g for endpoints of intervals from \mathcal{A} we
257 can compute all such values of function g in $|\mathcal{A}| \log \log n$ time, sort all those pairs in linear
258 time (using radix sort), and then define function $g'(X)$ as a rank of a pair in obtained sorted
259 sequences.

260 Finally for each interval $P = [X, Y]$, we define $f(P) = (g'(X), g'(Y))$. ◀

261 Let $\#$ be a character that is greater than all the letters in Σ .

262 ► **Observation 15.** *Assume that P and Q are factors of W . Then P is a prefix of Q iff
263 $Q \in [P, P\#]$ and a suffix of Q iff $Q^R \in [P^R, P^R\#]$.*

264 As suggested in the above observation, in factor intervals we will also use, as right
265 endpoints, strings of the form $F\#$ where F is a factor of W . In order to transform them
266 into integer intervals with Lemma 14, we will compute the maximum factor of W that starts
267 with F .

268 ► **Lemma 16.** *Let W be a string of length n . After $\mathcal{O}(n)$ -time preprocessing, given a fragment
269 x of W , one can compute the lexicographically minimal/maximal factor of W that starts with
270 a factor matching x in $\mathcal{O}(\log \log n)$ time.*

271 **Proof.** First we compute in $\mathcal{O}(n)$ time, the suffix tree of string W . Additionally we add to
272 each explicit node the index of the lexicographically minimal/maximal suffix from its subtree.
273 This can be easily done in $\mathcal{O}(n)$ time by traversing the suffix tree. The last ingredient of our
274 solution is preprocessing of the suffix tree for supporting weighted ancestor queries. Using
275 solution from Fact 13 the preprocessing requires $\mathcal{O}(n)$ time and each weighted ancestor query
276 can be answered in $\mathcal{O}(\log \log n)$ time.

277 Given fragment x of W starting at position i . We locate leaf v of suffix tree corresponding
278 to the suffix $W[i..n]$. Using weighted ancestor query we locate the highest ancestor u of
279 v , such that depth of u is greater than (or equal to) $|x|$. The resulting lexicographically
280 minimal/maximal factor of W (that starts with x) is the lexicographically minimal/maximal
281 suffix stored in the node u . ◀

282 7.3 Algorithm for 4-Families-Problem

283 We are now ready to show a solution to 4-FAMILIES-PROBLEM.

284 ► **Lemma 17.** *4-FAMILIES-PROBLEM can be solved in $\mathcal{O}(n + m \log^3 n)$ time and $\mathcal{O}(m \log^2 n)$
285 space.*

Proof. We construct two sets of factor 4-rectangles with weights, \mathcal{R}_1 and \mathcal{R}_2 , from \mathcal{F}_S and \mathcal{F}_T over the concatenation STS^RT^R . For every $(F_1, F_2, F_3, F_4) \in \mathcal{F}_S$, we add a factor rectangle

$$\{F_1^R\} \times [F_2, F_2\#] \times [F_3^R, F_3^R\#] \times \{F_4\} \text{ with weight } |F_2| + |F_3|$$

to \mathcal{R}_1 . For every $(F'_1, F'_2, F'_3, F'_4) \in \mathcal{F}_T$, we add a factor rectangle

$$[(F'_1)^R, (F'_1)^R\#] \times \{F'_2\} \times \{(F'_3)^R\} \times [F'_4, F'_4\#] \text{ with weight } |F'_1| + |F'_4|$$

286 to \mathcal{R}_2 . By Observation 15, the solution of 4-FAMILIES-PROBLEM corresponds to two
287 intersecting 4-rectangles $R_1 \in \mathcal{R}_1$ and $R_2 \in \mathcal{R}_2$ with maximum total weight.

288 Lemma 16 can be used to transform strings $X\#$ to factors of STS^RT^R . Then we apply
289 Lemma 14 to transform factor 4-rectangles to standard 4-rectangles, dimension per dimension.
290 Finally, we use Fact 12 to conclude. ◀

291 As a consequence of Propositions 7, 10, and 11 applied for all a, b and the above lemma we
292 obtain the main result.

293 ▶ **Theorem 18 (Main result).** *The LCCF problem can be solved in $\mathcal{O}(n \log^6 n)$ time by a*
294 *randomized (Las Vegas) algorithm.*

295 8 Conclusions

296 We have presented an $\mathcal{O}(n \log^6 n)$ -time randomized algorithm for computing the Longest
297 Common Circular Factor (LCCF) of two strings of length n . Let us recall that the Longest
298 Common Factor (LCF) of two strings can be computed in $\mathcal{O}(n)$ time. We leave an open
299 question if the LCCF problem can also be solved in linear time and if there exists a similarly
300 efficient deterministic algorithm that solves this problem.

301 — References —

- 302 **1** Amihood Amir, Gad M. Landau, Moshe Lewenstein, and Dina Sokol. Dynamic text and
303 static pattern matching. *ACM Trans. Algorithms*, 3(2):19, 2007. URL: [https://doi.org/](https://doi.org/10.1145/1240233.1240242)
304 [10.1145/1240233.1240242](https://doi.org/10.1145/1240233.1240242), doi:10.1145/1240233.1240242.
- 305 **2** Alberto Apostolico, Maxime Crochemore, Martin Farach-Colton, Zvi Galil, and
306 S. Muthukrishnan. 40 years of suffix trees. *Commun. ACM*, 59(4):66–73, 2016. URL:
307 <https://doi.org/10.1145/2810036>, doi:10.1145/2810036.
- 308 **3** Tanver Athar, Carl Barton, Widmer Bland, Jia Gao, Costas S. Iliopoulos, Chang Liu,
309 and Solon P. Pissis. Fast circular dictionary-matching algorithm. *Mathematical Struc-*
310 *tures in Computer Science*, 27(2):143–156, 2017. URL: [https://doi.org/10.1017/](https://doi.org/10.1017/S0960129515000134)
311 [S0960129515000134](https://doi.org/10.1017/S0960129515000134), doi:10.1017/S0960129515000134.
- 312 **4** Md. Aashikur Rahman Azim, Costas S. Iliopoulos, Mohammad Sohel Rahman, and
313 M. Samiruzzaman. A fast and lightweight filter-based algorithm for circular pattern
314 matching. In Pierre Baldi and Wei Wang, editors, *Proceedings of the 5th ACM Confer-*
315 *ence on Bioinformatics, Computational Biology, and Health Informatics, BCB '14, New-*
316 *port Beach, California, USA, September 20-23, 2014*, pages 621–622. ACM, 2014. URL:
317 <https://doi.org/10.1145/2649387.2660804>, doi:10.1145/2649387.2660804.
- 318 **5** Hideo Bannai, Tomohiro I, Shunsuke Inenaga, Yuto Nakashima, Masayuki Takeda, and
319 Kazuya Tsuruta. The "runs" theorem. *SIAM J. Comput.*, 46(5):1501–1514, 2017. URL:
320 <https://doi.org/10.1137/15M1011032>, doi:10.1137/15M1011032.

- 321 **6** Carl Barton, Costas S. Iliopoulos, and Solon P. Pissis. Fast algorithms for approximate
322 circular string matching. *Algorithms for Molecular Biology*, 9:9, 2014. URL: <https://doi.org/10.1186/1748-7188-9-9>, doi:10.1186/1748-7188-9-9.
- 323
- 324 **7** Carl Barton, Costas S. Iliopoulos, and Solon P. Pissis. Average-case optimal approximate
325 circular string matching. In Adrian-Horia Dediu, Enrico Formenti, Carlos Martín-Vide,
326 and Bianca Truthe, editors, *Language and Automata Theory and Applications - 9th International Conference, LATA 2015, Nice, France, March 2-6, 2015, Proceedings*, volume
327 8977 of *Lecture Notes in Computer Science*, pages 85–96. Springer, 2015. URL: https://doi.org/10.1007/978-3-319-15579-1_6, doi:10.1007/978-3-319-15579-1_6.
- 328
- 329
- 330 **8** Domenico Cantone, Simone Faro, and Arianna Pavone. Sequence searching allowing for
331 non-overlapping adjacent unbalanced translocations. *CoRR*, abs/1812.00421, 2018. URL:
332 <http://arxiv.org/abs/1812.00421>, arXiv:1812.00421.
- 333
- 334 **9** Kuei-Hao Chen, Guan-Shieng Huang, and Richard Chia-Tung Lee. Exact circular pattern
335 matching using the bndm algorithm. In *Proceedings of the 28th Workshop on Combinatorial Mathematics and Computation Theory*, pages 152–161. National Penghu University of
336 Science and Technology, Penghu, 2011.
- 337
- 338 **10** Kuei-Hao Chen, Guan-Shieng Huang, and Richard Chia-Tung Lee. Exact circular pattern
339 matching using bit-parallelism and q-gram technique. In *Proceedings of the 29th Workshop on Combinatorial Mathematics and Computation Theory*, pages 18–27. National Taipei
340 College of Business, Institute of Information and Decision Sciences, Taipei, 2012.
- 341
- 342 **11** Kuei-Hao Chen, Guan-Shieng Huang, and Richard Chia-Tung Lee. Bit-parallel algorithms
343 for exact circular string matching. *Comput. J.*, 57:731–743, 2013. doi:10.1093/comjnl/bxt023.
- 344
- 345 **12** Da-Jung Cho, Yo-Sub Han, and Hwee Kim. Alignment with non-overlapping inversions
346 and translocations on two strings. *Theor. Comput. Sci.*, 575:90–101, 2015. URL: <https://doi.org/10.1016/j.tcs.2014.10.036>, doi:10.1016/j.tcs.2014.10.036.
- 347
- 348 **13** Maxime Crochemore, Costas S. Iliopoulos, Marcin Kubica, Jakub Radoszewski, Wojciech
349 Rytter, and Tomasz Walen. Extracting powers and periods in a word from its runs structure. *Theor. Comput. Sci.*, 521:29–41, 2014. URL: <https://doi.org/10.1016/j.tcs.2013.11.018>, doi:10.1016/j.tcs.2013.11.018.
- 350
- 351 **14** Maxime Crochemore and Wojciech Rytter. *Jewels of Stringology*. World Scientific, 2003.
352 doi:10.1142/4838.
- 353
- 354 **15** Herbert Edelsbrunner. A new approach to rectangle intersections part i. *International Journal of Computer Mathematics*, 13:209–219, 1983. doi:10.1080/00207168308803364.
- 355
- 356 **16** Kimmo Fredriksson and Szymon Grabowski. Average-optimal string matching. *J. Discrete Algorithms*, 7(4):579–594, 2009. URL: <https://doi.org/10.1016/j.jda.2008.09.001>, doi:10.1016/j.jda.2008.09.001.
- 357
- 358 **17** Kimmo Fredriksson and Gonzalo Navarro. Average-optimal single and multiple approximate
359 string matching. *ACM Journal of Experimental Algorithmics*, 9(1.4):1–47, 2004. URL:
360 <https://doi.org/10.1145/1005813.1041513>, doi:10.1145/1005813.1041513.
- 361
- 362 **18** Szymon Grabowski, Tomasz Kociumaka, and Jakub Radoszewski. On abelian longest
363 common factor with and without RLE. *Fundam. Inform.*, 163(3):225–244, 2018. URL:
<https://doi.org/10.3233/FI-2018-1740>, doi:10.3233/FI-2018-1740.
- 364
- 365 **19** Tommi Hirvola and Jorma Tarhio. Bit-parallel approximate matching of circular strings
366 with k mismatches. *ACM Journal of Experimental Algorithmics*, 22, 2017. URL: <https://doi.org/10.1145/3129536>, doi:10.1145/3129536.
- 367
- 368 **20** Costas S. Iliopoulos, Solon P. Pissis, and M. Sohel Rahman. Searching and indexing circular
369 patterns. In Mourad Elloumi, editor, *Algorithms for Next-Generation Sequencing Data, Techniques, Approaches, and Applications.*, pages 77–90. Springer, 2017. URL: https://doi.org/10.1007/978-3-319-59826-0_3, doi:10.1007/978-3-319-59826-0_3.
- 370

- 371 **21** Costas S. Iliopoulos and M. Sohel Rahman. Indexing circular patterns. In Shin-Ichi Nakano
372 and Md. Saidur Rahman, editors, *WALCOM: Algorithms and Computation, Second Inter-*
373 *national Workshop, WALCOM 2008, Dhaka, Bangladesh, February 7-8, 2008.*, volume
374 4921 of *Lecture Notes in Computer Science*, pages 46–57. Springer, 2008. URL: https://doi.org/10.1007/978-3-540-77891-2_5, doi:10.1007/978-3-540-77891-2_5.
- 375
376 **22** Tomasz Kociumaka. *Efficient Data Structures for Internal Queries in Texts*. PhD thesis,
377 University of Warsaw, October 2018. URL: [https://www.mimuw.edu.pl/~kociumaka/](https://www.mimuw.edu.pl/~kociumaka/files/phd.pdf)
378 [files/phd.pdf](https://www.mimuw.edu.pl/~kociumaka/files/phd.pdf).
- 379 **23** Tomasz Kociumaka, Jakub Radoszewski, Wojciech Rytter, and Tomasz Walen. Internal
380 pattern matching queries in a text and applications. In Piotr Indyk, editor, *Proceedings*
381 *of the Twenty-Sixth Annual ACM-SIAM Symposium on Discrete Algorithms, SODA 2015,*
382 *San Diego, CA, USA, January 4-6, 2015*, pages 532–551. SIAM, 2015. URL: <https://doi.org/10.1137/1.9781611973730.36>, doi:10.1137/1.9781611973730.36.
- 383
384 **24** Jie Lin and Donald A. Adjeroh. All-against-all circular pattern matching. *Comput. J.*,
385 55(7):897–906, 2012. URL: <https://doi.org/10.1093/comjnl/bxr126>, doi:10.1093/
386 comjnl/bxr126.
- 387 **25** Jie Lin, Yue Jiang, and Don Adjeroh. Circular pattern discovery. *Comput. J.*, 58(5):1061–
388 1073, 2015. URL: <https://doi.org/10.1093/comjnl/bxu009>, doi:10.1093/comjnl/
389 bxu009.
- 390 **26** H. Ogiwara, T. Kohno, H. Nakanishi, K. Nagayama, M. Sato, and J. Yokota. Unbalanced
391 translocation, a major chromosome alteration causing loss of heterozygosity in human lung
392 cancer. *Oncogene*, 27(35):4788–4797, 2008. doi:10.1038/onc.2008.113.
- 393 **27** Robert Susik, Szymon Grabowski, and Sebastian Deorowicz. Fast and simple circu-
394 lar pattern matching. In Aleksandra Gruca, Tadeusz Czachórski, and Stanislaw Koziel-
395 ski, editors, *Man-Machine Interactions 3, Proceedings of the 3rd International Con-*
396 *ference on Man-Machine Interactions, ICMMI 2013, Brenna, Poland, October 22-25,*
397 *2013*, volume 242 of *Advances in Intelligent Systems and Computing*, pages 537–544.
398 Springer, 2013. URL: https://doi.org/10.1007/978-3-319-02309-0_59, doi:10.1007/
399 978-3-319-02309-0_59.
- 400 **28** Dorothy Warburton. De novo balanced chromosome rearrangements and extra marker
401 chromosomes identified at prenatal diagnosis: clinical significance and distribution of break-
402 points. *Am. J. Hum. Genet.*, 49(5):995–1013, 1991. URL: [https://www.ncbi.nlm.nih.](https://www.ncbi.nlm.nih.gov/pubmed/1928105)
403 [gov/pubmed/1928105](https://www.ncbi.nlm.nih.gov/pubmed/1928105).
- 404 **29** Brooke Weckselblatt, Karen E. Hermetz, and M. Katharine Rudd. Unbalanced transloca-
405 tions arise from diverse mutational mechanisms including chromothripsis. *Genome Res.*,
406 25(7):937–947, 2015. doi:10.1101/gr.191247.115.