Sorting by Multi-cut Rearrangements

Laurent Bulteau[1], Guillaume Fertin[2(✉)], Géraldine Jean[2], and Christian Komusiewicz[3]

[1] LIGM, CNRS, Univ Gustave Eiffel, ESIEE Paris, 77454 Marne-la-Vallée, France
laurent.bulteau@univ-eiffel.fr
[2] Université de Nantes, CNRS, LS2N, 44000 Nantes, France
guillaume.fertin@univ-nantes.fr, geraldine.jean@univ-nantes.fr
[3] Fachbereich für Mathematik und Informatik, Philipps-Universität Marburg, Marburg, Germany
komusiewicz@informatik.uni-marburg.de

Abstract. Let S be a string built on some alphabet Σ. A *multi-cut rearrangement* of S is a string S' obtained from S by an operation called *k-cut rearrangement*, that consists in (1) cutting S at a given number k of places in S, making S the concatenated string $X_1 \cdot X_2 \cdot X_3 \dots X_k \cdot X_{k+1}$, where X_1 and X_{k+1} are possibly empty, and (2) rearranging the X_is so as to obtain $S' = X_{\pi(1)} \cdot X_{\pi(2)} \cdot X_{\pi(3)} \dots X_{\pi(k+1)}$, π being a permutation on $1, 2 \dots k+1$ satisfying $\pi(1) = 1$ and $\pi(k+1) = k+1$. Given two strings S and T built on the same multiset of characters from Σ, the Sorting by Multi-cut Rearrangements (SMCR) problem asks whether a given number ℓ of k-cut rearrangements suffices to transform S into T. The SMCR problem generalizes and thus encompasses several classical genomic rearrangements problems, such as Sorting by Transpositions and Sorting by Block Interchanges. It may also model *chromoanagenesis*, a recently discovered phenomenon consisting in massive simultaneous rearrangements. In this paper, we study the SMCR problem from an algorithmic complexity viewpoint, and provide, depending on the respective values of k and ℓ, polynomial-time algorithms as well as NP-hardness, FPT-algorithms, W[1]-hardness and approximation results, either in the general case or when S and T are permutations.

1 Introduction

Genome rearrangements refer to large-scale evolutionary events that affect the genome of a species. They include among others reversals [1], transpositions [2], and block interchanges [5]; see also [9] for a full description. Compared to small-scale evolutionary events such as insertion, deletion or substitution of single DNA nucleotides, they are considered to be rare and, until recently, were assumed to happen one after the other. In the recent literature, however, a new type of event,

GF was partially supported by the PHC Procope program 17746PC
GJ was partially supported by the PHC Procope program 17746PC
CK was partially supported by the DAAD Procope program 57317050.

T. Bureš et al. (Eds.): SOFSEM 2021, LNCS 12607, pp. 593–607, 2021.
https://doi.org/10.1007/978-3-030-67731-2_43

called *chromoanagenesis*, has been shown to occur in genomes [12,13]. The term chromoanagenesis subsumes different types of rearrangements (namely, chromothripsis, chromoanasynthesis and chromoplexy) whose common ground is the following: in a single event, the genome is cut into many blocks, and then rearranged. As stated by Pellestor and Gatinois [12], these are "massive chromosomal rearrangements arising during single chaotic cellular events". Chromoanagenesis, and notably chromothripsis, is suspected to play a role in cancer and congenital diseases [13]. In this paper, we introduce a new model for genome rearrangements that is general enough to encompass most of the previously described genome rearrangements [9] as well as chromoanagenesis. Our goal here is to study its properties in terms of computational complexity.

Notation. Given an alphabet Σ, we say that two strings $S \in \Sigma^*$ and $T \in \Sigma^*$ are *balanced* if S and T are built on the same multiset of characters—in other words, each character in S also appears in T in the same number of occurrences. We denote by $|S|$ the length of a string S. Unless otherwise stated, we assume that $|S| = |T| = n$. We denote by S_i, $1 \leq i \leq n$, the i-th character of S. Given a string S in Σ^*, we denote by d the maximum number of occurrences of any character of Σ in S. In the specific case where $d = 1$ (i.e. when S and T are permutations), and for any $0 \leq i \leq n$, we say that there is a *breakpoint* at position i in S (or, equivalently, that (S_i, S_{i+1}) is a breakpoint) if the two consecutive characters S_i and S_{i+1} are not consecutive in T. For the specific cases $i = 0$ and $i = n$, we artificially set $S_0 = T_0 = \alpha_0$ and $S_{n+1} = T_{n+1} = \alpha_{n+1}$ where $\alpha_0 \notin \Sigma$ and $\alpha_{n+1} \notin \Sigma$. Thus, there is a breakpoint at position 0 (resp. n) in S whenever $S_1 \neq T_1$ ($S_n \neq T_n$). We also denote by $b(S, T)$ the number of *breakpoints* in S with respect to T. If (S_i, S_{i+1}) is not a breakpoint, we say that it is an *adjacency*.

Definition 1. *Given a string $S \in \Sigma^*$ and an integer k, a* k-cut rearrangement *of S is an operation consisting in the two following steps: (1) cut S at k locations (thus S can be written as the concatenation of $k+1$ strings, i.e. $S = X_1 \cdot X_2 \cdot X_3 \dots X_{k+1}$, where each X_i is possibly empty, and where a cut occurs between X_i and X_{i+1}, $1 \leq i \leq k$) and (2) rearrange (i.e., permute) the X_is so as to obtain $S' = X_{\pi(1)} \cdot X_{\pi(2)} \cdot X_{\pi(3)} \dots X_{\pi(k+1)}$, π being a permutation on the elements $1, 2 \dots k+1$ such that $\pi(1) = 1$ and $\pi(k+1) = k+1$. Each of the X_is considered in a given k-cut rearrangement will be called a* block.

Note that, although a k-cut rearrangement has been defined as a cut along the string at k locations, it is always possible, if necessary, to perform only $k' \leq k$ cuts at a given step—thus mimicking a k-cut rearrangement while actually realizing a k'-cut rearrangement—by cutting several times at the same location. The case where X_1 (resp. X_{k+1}) is empty corresponds to the case where the leftmost (resp. rightmost) block of S is moved to obtain S', otherwise X_1 (resp. X_{k+1}) remains unmoved in S'. Note that, in this model, each of the blocks X_is can only be permuted, thus no reversal of an X_i is allowed, and therefore the strings we consider are always unsigned. In this paper, we study the following problem.

Sorting by Multi-cut Rearrangements (SMCR)
Instance: Two balanced strings S and T, two integers ℓ and k.
Question: Is there a sequence of at most ℓ many k-cut rearrangements that transforms S into T?

For convenience, we may also refer to the SMCR problem with parameters k and ℓ as the $(k, \ell)-$SMCR problem. Our goal in this paper is to provide algorithmic results regarding SMCR. The computational complexity of SMCR highly depends on whether we set bounds on k and ℓ: depending on applications, they can either be fixed constants (and in that case algorithms running in e.g. $O(n^k)$ are acceptable), parameters (unbounded, but far smaller than n, then algorithms in $f(k) \cdot \text{poly}(n)$—that is, Fixed-Parameter Tractable (or FPT) algorithms [7,8] —would be relevant even for fast-growing functions f), or parts of the input (i.e. unbounded, and in that case only polynomial-time algorithms on n and k are relevant). Hence, we will consider each of these cases for both ℓ and k. For this study, we will consider separately the case of strings (i.e., $d > 1$ both in S and T) from the case of permutations (i.e., $d = 1$ both in S and T), in Sects. 2 and 3, respectively.

Basic Observations. Both in permutations and strings, the cases $k = 1$ and $k = 2$ are trivial, since they do not allow to move any block, and thus we are in presence of a Yes-instance iff $S = T$.

Additionally, the SMCR problem is a natural generalization and extension of a certain number of problems that have already been defined and studied in the literature before, as described hereafter.

When $k = 3$, each k-cut rearrangement is necessarily a transposition of blocks X_2 and X_3. Thus SMCR in that case is equivalent to the Sorting by Transpositions problem [2], for which we know it is NP-hard, even if S and T are permutations [3].

When $k = 4$, each k-cut rearrangement allows to move two blocks among X_2, X_3 and X_4, which exactly corresponds to the Sorting by Block Interchange problem. This problem is known to be in P for permutations [5] and NP-hard for strings (an NP-hardness proof for binary strings is given in [6], Theorem 5.7.2).

When $\ell = 1$, the SMCR problem comes down to deciding whether k cuts are sufficient to rearrange S into T in one atomic move (i.e., one k-cut rearrangement). In permutations, the problem is trivially solved by counting the number $b(S, T)$ of breakpoints between S and T, since we have a Yes-instance iff $b(S, T) \leq k$. In strings, the SMCR problem resembles the Minimum Common String Partition problem [11], as will be discussed in Theorems 3 and 4.

When ℓ and k are constant, SMCR is trivially polynomial-time solvable, since a brute-force algorithm, exhaustively testing all possible k-cut rearrangements at each of the ℓ authorized moves, has a running time of $O(n^{k\ell+1})$—the additional n factor being needed to verify that the result corresponds to string T.

It is also natural to wonder whether (k, ℓ)-SMCR and $(k\ell, 1)$-SMCR are equivalent. It can be easily seen that a YES-instance for (k, ℓ)-SMCR is also a YES-instance for $(k\ell, 1)$-SMCR: it suffices for this to aggregate all cuts from the (k, ℓ)-SMCR solution (of which there are at most $k\ell$), and rearrange accordingly. However, the reverse (i.e. from $(k\ell, 1)$-SMCR to (k, ℓ)-SMCR) is not always true. For example, take $S = \texttt{afedcbg}$, $T = \texttt{abcdefg}$, $k = 3$, and $\ell = 2$: this is a YES-instance for $(6, 1)$-SMCR, whereas it is a NO-instance for $(3, 2)$-SMCR. Indeed, in this instance the number $b(S, T)$ of breakpoints is equal to 6. Thus, in the former case, the 6 following cuts (symbolized as vertical segments) in $S = \texttt{a|f|e|d|c|b|g}$ suffice to obtain T after a single 6-cut rearrangement. In the latter case, every 3-cut rearrangement is a transposition, and in this instance no transposition can decrease $b(S, T)$ by 3. Thus at least three 3-cut rearrangements are necessary to transform S into T.

2 Sorting by Multi-cut Rearrangements in Strings

In this section, we provide algorithmic results concerning the SORTING BY MULTI-CUT REARRANGEMENTS problem, in the general case where S and T are strings. Our results are summarized in Table 1.

Table 1. Summary of the results for SORTING BY MULTI-CUT REARRANGEMENTS in strings. d is the maximum number of occurrences of a character in the input string S.

<table>
<tr><th>ℓ \ k</th><th>$O(1)$</th><th>parameter</th><th>part of the input</th></tr>
<tr><td>1</td><td rowspan="2">P</td><td>FPT (Thm 4)</td><td rowspan="3">NP-hard:
for $\ell = 1$ even when $d = 2$ (Thm 3)
for any fixed $\ell \geq 1$ (Thm 2)</td></tr>
<tr><td>$O(1)$</td><td rowspan="2">?</td></tr>
<tr><td>parameter</td></tr>
<tr><td>part of the input</td><td colspan="3">for any $k \geq 5$ even in k-ary strings (Thm 1)
for $k = 3, 4$ even in binary strings [3,6]</td></tr>
</table>

As mentioned in the previous section, we know that SMCR is NP-hard in binary strings for $k = 3, 4$. In the following theorem, we extend this result to any value of k, however in larger alphabet strings.

Theorem 1. SMCR *is NP-hard for any fixed $k \geq 5$, even in k-ary strings.*

Proof. We reduce the NP-hard 3-PARTITION problem in which the input is a set $\mathcal{A}$ of integers and an integer B, and the question is whether $\mathcal{A}$ can be partitioned into triples such that the integers of each triple sum up to B. Given an instance of 3-PARTITION $(\mathcal{A}, B)$ with $\mathcal{A} = \{a_1, a_2, \ldots, a_{3m}\}$ and $mB = \sum_{i=1}^{3m} a_i$, we construct an instance of SMCR for any fixed $k \geq 5$ as follows. For ease of presentation, we assume that each a_i is a multiple of 4 m and that $\frac{B}{4} < a_i < \frac{B}{2}$,

so that we have the following property: If for some subset I of $\{1,\dots,3m\}$ and some δ with $0 \le \delta \le 4\,\mathrm{m}$ we have $\sum_{i\in I} a_i + \delta = B+4$, then $\sum_{i\in I} a_i = B$, $\delta = 4$, and $|I| = 3$. We use a size-k alphabet $\{0,1,\dots,k-1\}$, we denote by X the string $k{-}1{\cdot}k{-}2{\cdot}\dots 3$, and by X' the reverse of X, i.e. $3{\cdot}4{\cdot}\dots{\cdot}k{-}1$ (note that X and X' have length $k-3 \ge 2$). We define $S := 10^{a_1}10^{a_2}\dots 10^{a_{3m}}1(20X0X0X0)^m2$ and $T := (1X')^{3m}1(20^{B+4})^m2$, and set $\ell = 3\,\mathrm{m}$. This completes the construction. Before proving its correctness, we group the adjacencies of the strings S and T based on whether the adjacencies of the two involved characters are in excess in S, in T, or equal in both strings.

- *Group 1* contains the adjacencies $(0,1),(1,0),(0,k{-}1),(k{-}1,k{-}2),\dots,(4,3)$, and $(3,0)$ which each occur $3\,\mathrm{m}$ times in S and which do not occur in T.
- *Group 2* contains the adjacencies $(0,0)$, which occur $Bm-3\,\mathrm{m}$ times in S and $Bm+3\,\mathrm{m}$ times in T, and the adjacencies $(1,3),(3,4),\dots,(k{-}2,k{-}1),(k{-},1)$ which do not occur in S, and occur $3\,\mathrm{m}$ times each in T.
- *Group 3* contains the adjacencies $(0,2)$ and $(2,0)$ which each occur m times in S and in T.

There are no further adjacencies in S or T. To show the correctness of the reduction we show that $(\mathcal{A},B)$ is a Yes-instance of 3-Partition iff there exists a sequence of at most $\ell = 3\,\mathrm{m}$ many k-cut rearrangements transforming S into T.

($\Rightarrow$) Pick a solution of 3-Partition. For each triple (a_i,a_j,a_p) of the solution, choose a unique substring $20X0X0X0$ of S and perform the following three k-cut rearrangements: First, cut S around 0^{a_i} and around the first copy of X in the chosen subsequence, and cut at every position inside X. Observe that the number of cuts is exactly k. Now reverse X into X' and exchange 0^{a_i} and X'. Perform a similar k-cut rearrangement with a_j and the second occurrence of X and with a_p and the third occurrence of X in the selected substring. The selected substring is transformed into $200^{a_i}00^{a_j}00^{a_k}0 = 20^{B+4}$ and since each string 0^{a_i} is replaced by X', the first part of the string is $(1X')^{3m}$. Hence, the string obtained by the $3\,\mathrm{m}$ many k-cut rearrangements described above is T.

($\Leftarrow$) There are altogether $6\,\mathrm{m}$+(k-2)$3\,\mathrm{m} = 3\,\mathrm{km}$ adjacencies in Group 1 which are in excess in S, and $3\,\mathrm{km}$ adjacencies in Group 2 which are in excess in T. Since $\ell = 3\,\mathrm{m}$, each k-cut rearrangement cuts k adjacencies in Group 1 (and no adjacency in Group 2 or 3). In particular, no 00 adjacency may be cut in a feasible solution, so each subsequence 0^{B+4} in T is obtained by concatenating a number of strips of the form 0^{a_i} as well as some number δ of 0 singletons. Since S has $4\,\mathrm{m}$ of these singleton 0s, we have $0 \le \delta \le 4\,\mathrm{m}$. By the constraint on the values of a_i, each subsequence 0^{B+4} in T contains four singletons from S and three substrings 0^{a_i} of S whose lengths sum to B. Thus, the m substrings 0^{B+4} in T correspond to a partition of $\mathcal{A}$ into m sets of three integers whose values sum up to B. □

Theorem 2. SMCR *is NP-hard for any fixed* ℓ.

Proof. The reduction being very similar to the one of Theorem 1, we only highlight the differences to have a fixed ℓ instead of fixed k. First assume that m is a

multiple of ℓ (add up to ℓ triples of dummy elements otherwise), and let $k = \frac{15m}{\ell}$. Note that k is a multiple of 5. The reduction is the same as above with a size-5 alphabet. In other words, we have $X = 43$ and $X' = 34$.

In the forward direction, use the described scenario using 5-cut rearrangements, but combine a series of $k/5$ such rearrangements into a single k-cut rearrangement, as described at the end of Sect. 1. This gives a total of $\frac{3m}{k/5} = \ell$ many k-cut rearrangements sorting S into T. In the reverse direction, the same breakpoint count holds, namely 3 k'm adjacencies need to be broken using ℓ k-cut rearrangements, with $\ell k = 3$ k'm. So again no 00 adjacency may be broken, and by the same argument, we obtain a valid 3-partition of $\mathcal{A}$. □

The previous theorem shows NP-hardness of SMCR for any fixed ℓ. However, a stronger result can be obtained in the specific case $\ell = 1$.

Theorem 3. SMCR *is NP-hard when* $\ell = 1$*, even when* $d = 2$*.*

Proof. The proof is obtained by reduction from the Minimum Common String Partition problem, which has been proved to be NP-hard in strings, even when $d = 2$ [11]. Recall that the decision version of MCSP asks, given two balanced strings S and T, and an integer p, whether S can be written as the concatenation of p blocks $S = X_1 \cdot X_2 \dots X_{p-1} \cdot X_p$ and T can be written as $T = X_{\pi(1)} \cdot X_{\pi(2)} \dots X_{\pi(p-1)} \cdot X_{\pi(p)}$, where π is a permutation of $1, 2 \dots p$. Note that here we may have $\pi(1) = 1$ and/or $\pi(p) = p$.

Given an instance (S, T, p) of MCSP, we build an instance (S', T', k, ℓ) of SMCRby setting $S' = x \cdot S$, $T' = T \cdot x$ (with $x \notin \Sigma$), $k = p + 2$ and $\ell = 1$. Clearly, if (S, T, p) is a Yes-instance for MCSP, then $(S', T', p + 2, 1)$ is a Yes-instance for SMCR: the MCSP solution uses $p - 1$ cuts, to which we add one before x, one after x, and one after S_n for solving SMCR. Conversely, if SMCR is a Yes-instance for SMCR, and since x occurs only once in S', then $S'_1 = x$, and thus 2 cuts are used to "isolate" x from S'. Besides, since T' ends with x, there must exist a cut after the last character of S'. Hence, since $S' = x \cdot S$, at most $k - 3 = p - 1$ cuts are used strictly within S, which in turns means that S has been decomposed in p blocks, which can be rearranged so as to obtain T since $\ell = 1$. Thus, (S, T, p) is a Yes-instance for MCSP. □

Note that MCSP has been proved to be in FPT with respect to the size of the solution [4]. It can be seen that this result can be adapted for the SMCR problem in the case $\ell = 1$.

Theorem 4. *When* $\ell = 1$*,* SMCR *is FPT with respect to parameter* k*.*

Proof. Assuming $S \neq T$, let A (resp. B) be the length of the longest common prefix (resp. suffix) of S and T. For $0 \leq a \leq A$ and $0 \leq b \leq B$, let $S_{a,b}, T_{a,b}$ be the strings obtained from S, T by removing the first a and last b characters. Then T can be obtained from S by one k-cut rearrangement if and only if, for some pair (a, b), $S_{a,b}$ and $T_{a,b}$ admit a common string partition into $k-1$ blocks. Indeed, this is easy to verify by matching the limits of the blocks in MCSP

(including at the end of the strings) with the cuts of the rearrangement. So SMCR when $\ell = 1$ can be solved using $O(n^2)$ calls to MCSP with parameter $k-1$, each with a different pair (a, b), which itself is FPT for k [4]. □

Note that it is not sufficient to check only with the longest common prefix and suffix (i.e. $S_{A,B}$ and $T_{A,B}$), as can be seen in the following example, where S can be transformed into T via one 3-cut rearrangement, $A = B = 2$, but only $S_{1,1}$ and $T_{1,1}$ have a common partition into 2 blocks: $S =$ `a acb adb b` and $T =$ `a adb acb b`.

3 Sorting by Multi-cut Rearrangements in Permutations

In this section, we provide algorithmic results concerning the Sorting by Multi-cut Rearrangements problem, in the specific case where S and T are permutations. Our results are summarized in Table 2.

Table 2. Summary of the results for Sorting by Multi-cut Rearrangements in permutations. *existence of a 2-approximation algorithm for Opt-SMCR (Theorem 8).

<table>
<tr><th>ℓ \ k</th><th>3</th><th>4</th><th>$O(1)$</th><th>parameter</th><th>part of the input</th></tr>
<tr><td>1</td><td colspan="5" rowspan="2">P</td></tr>
<tr><td>$O(1)$</td></tr>
<tr><td>parameter</td><td>FPT (Thm 5)</td><td></td><td colspan="2">FPT (Thm 5)</td><td>W[1]-hard (Thm 6)</td></tr>
<tr><td>part of the input</td><td>NP-hard [3]</td><td></td><td colspan="3">NP-hard (Thm 7) *</td></tr>
</table>

Theorem 5. SMCR *in permutations is FPT with respect to parameter* $\ell + k$.

Proof. We obtain the fixed-parameter tractability result by using the following reduction rule: If there is a common adjacency (a, b) in S and T, then remove b from S and T. Before we show the correctness, observe that exhaustive application of the rule indeed gives the desired result: Any Yes-instance that is reduced exhaustively with respect to the above rule has $O(k\ell)$ letters: We must cut between every adjacency in S. Overall, we may create at most $2k\ell$ cuts via ℓ many k-cut rearrangements. Hence, if S has more than $2k\ell$ adjacencies, then (S, T) is a No-instance. Thus, after applying the rule exhaustively, we either know that the instance is a No-instance or we may solve it in $f(k, \ell)$ time by using a brute-force algorithm. Thus it remains to show correctness of the rule. Consider an instance consisting of the permutations S and T to which the rule is applied and let S' and T' denote the resulting instance. We show that (S, T) is a Yes-instance if and only if (S', T') is a Yes-instance.

($\Rightarrow$) If (S, T) is a Yes-instance, then there is a sequence of $\ell + 1$ permutations $(S = S_1, S_2, \ldots, S_{\ell+1} = T)$ such that S_{i+1} can be obtained from S_i via one

k-cut rearrangement. Removing b from S_i gives a sequence $(S'_1, S'_2, \ldots, S'_{\ell+1} = T)$ such that S'_{i+1} can be obtained from S'_i via one k-cut rearrangement.

($\Leftarrow$) If (S', T') is a YES-instance, then there is a sequence of $\ell + 1$ permutations $(S' = S'_1, S'_2, \ldots, S'_{\ell+1} = T')$ such that S'_{i+1} can be obtained from S'_i via one k-cut rearrangement. Replacing a by ab in each permutation S'_i gives a sequence $(S = S_1, S_2, \ldots, S_{\ell+1} = T)$ such that S'_{i+1} can be obtained from S'_i via one k-cut rearrangement. □

Theorem 6. SMCR *in permutations is W[1]-hard parameterized by* ℓ.

Proof. The proof is by reduction from UNARY BIN PACKING, whose instance is a multiset $\mathcal{A} = \{a_1, a_2, \ldots, a_n\}$ of integers encoded in unary, and two integers b and C. The goal is to decide whether one can partition $\mathcal{A}$ into b multisets $A_1, \ldots, A_b$, such that $\sum_{a_j \in A_i} a_j \leq C$, for each $1 \leq i \leq b$. This problem has been shown to be W[1]-hard [10] with respect to the number of multisets b, even when the sum of the elements $\sum_{i=1}^{n} a_i$ is equal to bC.

Take an instance I of UNARY BIN PACKING such that $\sum_{i=1}^{n} a_i = bC$. From I, we construct, in polynomial time, the following instance I^* of SMCR. We first define S as the following permutation of $[bC + 1]$:
$S := \mathbf{1}\, X^1\, (\mathbf{a_1 + 1})\, X^2\, (\mathbf{a_1 + a_2 + 1}) \ldots (\sum_{\mathbf{i=1}}^{\mathbf{n-1}} \mathbf{a_i + 1})\, X^n\, (\sum_{\mathbf{i=1}}^{\mathbf{n}} \mathbf{a_i + 1})$, where each X^i is the length-$(a_i - 1)$ decreasing sequence over $\{\sum_{j=1}^{i-1} a_j + 2, \ldots, \sum_{j=1}^{i} a_j\}$, that is, $X^i[k] := (\sum_{j=1}^{i} a_j) - (k - 1)$ for $1 \leq k \leq a_i - 1$. We set T to be the identity over the same alphabet $[bC + 1]$.

An element at position i in S is called an *anchor* if $S_i = i$ (in bold above). Since we want to transform S into the identity permutation, the anchors correspond to fixed points that are already well located. For any $1 \leq i \leq n$, the reversed sequence X^i is delimited by two anchors. We finally set $\ell = b$ and $k = C$. Each X^i has exactly a_i breakpoints: two at its extremities with the anchors and $a_i - 2$ internal ones. Since $\sum_{i=1}^{n} a_i = bC$, it can be seen that S contains exactly ℓk breakpoints. Since a k-cut rearrangement can remove at most k breakpoints, then at least ℓ such rearrangements are necessary to sort S. We now show that I is a YES-instance for UNARY BIN PACKING problem iff I^* is a YES-instance for SMCR.

($\Rightarrow$) Suppose I is a YES-instance for UNARY BIN PACKING. Thus there exists a partition $A_1 \ldots A_b$ of the multiset $\mathcal{A}$. To sort S, the k-cut rearrangements we apply consist in reversing the S^is. Note that in order to reverse a complete S^i corresponding to a given a_i, $1 \leq i \leq n$, we need exactly a_i cuts, e.g. to transform $\sigma = \ldots\, \mathbf{p + 1} | p + a_i | p + a_i - 1 | \ldots | p + 2 | \mathbf{p + a_i + 1} \ldots$ into $\rho = \ldots\, (\mathbf{p + 1})\, (p + 2) \ldots (p + a_i - 1)\, (p + a_i)\, (\mathbf{p + a_i + 1}) \ldots$ (where $p = \sum_{k=1}^{i-1} a_k$).

For any $1 \leq i \leq b$, we have $\sum_{a_j \in A_i} a_j = C$ and since $C = k$, we can define a k-cut rearrangement that consists in reversing the $X^{j_1}, X^{j_2}, \ldots, X^{j_p}$ corresponding to elements $a_{j_1}, a_{j_2}, \ldots, a_{j_p}$ of A_i. Since there are b such multisets and since $b = \ell$, ℓ such k-cut rearrangements are sufficient to sort S.

($\Leftarrow$) Suppose I^* is a YES-instance for SMCR. Since $b(S, T) = \ell k$, using ℓ k-cut rearrangements to sort π means that each rearrangement removes exactly

k breakpoints. It is only feasible if each X^i is reversed at once (i.e., during a single k-cut rearrangement) using exactly a_i cuts. Indeed, if we cut at $c_i < a_i$ places in X^i, we will be able to fix strictly less than c_i breakpoints and so the k-cut rearrangement in which the c_i cuts take place will not remove k breakpoints as expected. By following the moves during the sorting of S, it suffices to see which X^is are reversed within the same k-cut rearrangement. In that case, the sum of the corresponding a_is is equal to $k = C$ and, using $\ell = b$, such multisets of a_is provide a solution to Unary Bin Packing. □

Theorem 7. *For any $k \geq 5$,* SMCR *in permutations is NP-hard.*

This hardness proof is by reduction from Sorting By Transpositions on 3-cyclic permutations [3]. Intuitively, in such permutations, it is straightforward to identify triples of breakpoints, called 3-cycles, that should be solved together in a transposition, however the difficulty arises in selecting a correct order in which those 3-cycles should be solved. Our approach consists in extending these 3-cycles into k-cycles, such that any k-cut rearrangement solving the original cycle must solve all k breakpoints together, and still performs a simple transposition on the rest of the sequence (to this end, $k-3$ dummy elements are created in order to consume the extra blocks in k-cut rearrangements). We first recall the necessary definitions and properties for breakpoints and cyclic permutations, then show how to extend a single cycle by only two or three elements, and finally successively apply this method to extend *all* cycles to *any* size $k \geq 5$.

Breakpoints and Cycle Graph. For a permutation S of length n, we assume the alphabet of S is $\{1, \ldots, n\}$. We further write $S_0 = 0$ and $S_{n+1} = n + 1$. For a rearrangement r transforming S into S', we write $r(S) = S'$ and $r(S, T) = (S', T)$. The *cycle graph* $\mathcal{C}(S, T)$ of strings S and T is the graph over $n + 1$ vertices $\{0, \ldots n\}$ with arcs $T_j \to S_i$ if $T_{j+1} = S_{i+1}$. Every vertex has in-degree and out-degree 1, so the graph is a disjoint union of cycles. Self-loops are called *trivial cycles* (when seen as a cycle) or *adjacencies* (when seen as an arc), other arcs are *breakpoints*. An element (or vertex) x is an adjacency (resp. breakpoint) according to its outgoing arc (we use transparently the bijection between a vertex and its outgoing arc). A *k-cycle* is a cycle of length k. The *next breakpoint* of breakpoint $x \to y$ in $\mathcal{C}(S, T)$ is y (or equivalently, the outgoing arc of y). We write $\mathcal{C}_x(S, T)$ for the cycle of $\mathcal{C}(S, T)$ containing element x. A cycle graph is *k-cyclic* (and, by extension, a pair of sequences generating this cycle graph) if it contains only adjacencies and k-cycles. A rearrangement r applied to a permutation S *cuts* an element x, $0 \leq x \leq n$, if it cuts between $x = S_i$ and S_{i+1}. Furthermore, it *solves breakpoint* x if r cuts x and x is an adjacency in $r(S)$. It *solves a cycle* if it solves all breakpoints in it. We write $d_b(S, T)$ for the number of breakpoints of $\mathcal{C}(S, T)$. A k-cut rearrangement is *efficient* if it solves k breakpoints. A pair (S, T) is *k-efficiently sortable* if there exists a sequence of efficient k-cut rearrangements transforming S into T. The following is a trivial generalization of a well-known lower bound for the transposition distance.

Proposition 1. *A k-cut rearrangement may not solve more than k breakpoints, so S needs at least $\frac{d_b(S,T)}{k}$ k-cut rearrangements to be transformed into T. Furthermore, the bound is reached if and only if (S,T) is k-efficiently sortable.*

Proposition 2. *If r solves a breakpoint, it cuts the next breakpoint in the cycle graph.*

Proof. Let $x \to y$ be an arc of the cycle graph, and let x' be the successor of x in T as well as the successor of y in S. If r solves x, then r joins a block ending in x with a block starting in x', so x' is the first element of some block of r. Thus, y is the last element of some block of r, and r cuts the breakpoint y in $\mathcal{C}(S,T)$. □

Proposition 3. *If r is efficient, it solves a cycle iff it solves* any *breakpoint in it. Furthermore r solves all breakpoints in a union of cycles of total size k.*

Proof. If r is efficient, then it solves all breakpoints that it cuts (since it may not solve a breakpoint without cutting them, and it solves and cuts k breakpoints). By Proposition 2, if r solves a breakpoint in a cycle, then it must solve all subsequent arcs in the same cycle. Hence, r either solves all breakpoints of a cycle or none at all. The size constraint follows from the fact that all cycles are disjoint. □

Cycle C_1 is *tied* to another cycle C_2 through the pair of breakpoints (x,y) if x is in C_1, y is in C_2, the permutation S has $S_i = y$ and $S_{i+1} = x$ for some i, and T has $T_j = x$ and $T_j j + 1 = y$ for some y. A breakpoint is *without ties* if no cycle is tied to the cycle containing it.

Proposition 4. *If C_1 is tied to C_2, then any efficient rearrangement solving C_1 must also solve C_2.*

Proof. Let r be an efficient rearrangement solving C_1 and, in particular, x. Then r must place y after x in $r(S)$, although y is before x in S, so r must have a cut somewhere between y and x, i.e. just after y. So r cuts breakpoint y, and solves cycle C_2. □

One-cycle Extensions. Let (S,T) be a pair of permutations. Let x be a vertex of $\mathcal{C}(S,T)$ with the following properties (we say that x is *safe*): x is either an adjacency or a breakpoint without ties in a cycle of length $k_x \geq 3$, and all 2-cycles in $\mathcal{C}(S,T)$ are tied. The *p-extension of (S,T) on x*, with $p \in \{2,3\}$, denoted $\phi_x^p(S,T)$ is the pair (S',T') such that:

- For $p = 2$:
  $S' = (S_1, \ldots, S_i = x, n+2, S_{i+1}, \ldots, S_n, n+1)$ if x is an adjacency
  $S' = (S_1, \ldots, S_n, n+1, n+2)$ if x is a breakpoint
  $T' = (T_1, \ldots, T_j = x, n+2, T_{j+1} = S_{i+1}, \ldots T_n, n+1)$
- For $p = 3$:
  $S' = (S_1, \ldots, S_i = x, n+3, n+2, S_{i+1}, \ldots, S_n, n+1)$ if x is an adjacency
  $S' = (S_1, \ldots, S_n, n+1, n+2, n+3)$ if x is a breakpoint
  $T' = (T_1, \ldots, T_j = x, n+3, n+2, T_{j+1} = S_{i+1}, \ldots T_n, n+1)$

Lemma 1. *A p-extension on x has the following effects on the cycle graph:*

- *If x is an adjacency, it adds p trivial cycles.*
- *If x is a breakpoint and $p = 2$, it adds $n+1$ and $n+2$ to the cycle containing x.*
- *If x is a breakpoint and $p = 3$, it adds $n + 2$ to the cycle containing x and a 2-cycle $(n + 1, n + 3)$ tied to the one containing x.*

Other arcs and tied cycles are unchanged.

Proof. If x is an adjacency, the p-extension inserts elements $n+1$ to $n+p$ in both strings in the same order after x, and they are followed by the same element in both strings since x is an adjacency, so only trivial cycles are added.

Assume now that x is a breakpoint. Consider first an arc $T_j \to S_i$ with $T_j \neq x$ in $\mathcal{C}(S, T)$. Since no element is inserted after T_j or S_i, $T_j \to S_i$ also appears in $\mathcal{C}(S', T')$ (the case $i = j = n$ is particular, as $n + 1$ is explicitly introduced in both sequences, but it also yields the arc $T_j \to S_j$ in $\mathcal{C}(S', T')$). If a cycle is tied to another one through a pair (x, y) in S and (y, x) in T, these factors cannot be broken by the p-extension (since x is safe, no cycle can be tied to $\mathcal{C}_x(S, T)$), so it is still tied after the extension. Similarly, a non-tied cycle cannot become tied because of the extension.

It remains to describe arcs going out from $\{x, n+1, \ldots, n+p\}$. Let y be the head of the outgoing arc from x.

For j such that $T'_j = x$, we have $T'_{j+1} = n + p = S'_{n+p}$, so there exists an arc $x \to S'_{n+p-1} = n + p - 1$ in $\mathcal{C}(S', T')$ (note in particular that y no longer has its incoming arc $x \to y$).

For j such that $T'_j = n + 1$, we have $T'_{j+1} = n + p + 1 = S'_{n+p+1}$, so there exists an arc $n + 1 \to S'_{n+p} = n + p$ in $\mathcal{C}(S', T')$.

For $p = 3$ and j such that $T'_j = n+3$, we have $T'_{j+1} = n+2 = S'_{n+2}$, so there exists an arc $n + 3 \to S'_{n+1} = n + 1$ in $\mathcal{C}(S', T')$.

At this point, the out-going arcs for all vertices except $n + 2$ have been described, as well as in-coming arcs for all vertices except y, so the last remaining arc is $n + 2 \to y$.

Overall, for $p = 2$, arc $x \to y$ is replaced with the path $x \to n+1 \to n+2 \to y$. For $p = 3$, arc $x \to y$ is replaced with $x \to n+2 \to y$ and a 2-cycle $n+1 \leftrightarrow n+3$ is created. Note that this 2-cycle is tied to $\mathcal{C}_x(S', T')$ through $(n + 2, n + 3)$. □

We now show how efficient rearrangements can be adapted through extensions. Let r be a k-cut rearrangement of (S, T). We write $r' = \psi_x^p(r)$ for the k'-cut rearrangement of $(S', T') = \phi_x^p(S, T)$ defined as follows:

- If r does not cut x, then $k' = k$, r' cuts the same elements as r, and rearranges the blocks in the same order.
- If r cuts x, then $k' = k + p$, r' cuts the same elements as r as well as $n + 1$, $n + 2$ and $n + 3$ (when $p = 3$), and rearranges the blocks in the same way as r, with elements $n + 3$ (when $p = 3$) and $n + 2$ inserted after x.

The following two lemmas show how efficient rearrangements of (S, T) and those of $\phi_x(S, T)$ are related through ψ_x^p.

Lemma 2. *If r is an efficient k-cut rearrangement of (S,T), then $r' = \psi_x(r)$ is an efficient k'-rearrangement of $(S',T') = \phi_x^p(S,T)$. Furthermore $r'(S',T') = \phi_x^p(r(S,T))$.*

Proof. If r does not cut x, then $k' = k$ and r' solves in (S',T') exactly the same breakpoints as r, so it is efficient. Furthermore, all elements in $r'(S',T')$ and $\phi_x^p(r(S,T))$ are in the same order as in $r(S,T)$, except for $n+1,\dots,n+p$ which are inserted, in both case, at the end of S and in T as in T' (since r and r' do not edit the second string).

If r cuts x, r' furthermore solves breakpoints $n+1,\dots,n+p$, since it rearranges these elements in the same order as in T'. So it is an efficient rearrangement as well. Finally, all elements in $r'(S',T')$ and $\phi_x^p(r(S,T))$ are in the same order as in $r(S,T)$, except for $n+p,\dots,n+2$ (which are inserted after x in both strings) and $n+1$ (which is inserted as a last element). □

Lemma 3. *If r' is an efficient k'-rearrangement of $(S',T') = \phi_x^p(S,T)$ with $k' \in \{k_x, k_x+p\}$, then there exists an efficient k-cut rearrangement r of (S,T) such that $r' = \psi_x(r)$, where $k = k' - p = k_x$ if r' cuts x and $k = k'$ otherwise. Furthermore, $r'(S',T') = \phi_x^p(r(S,T))$.*

Proof. We build r from r' using the converse operations of Lemma 2: mimicking the cuts and reordering of r', but ignoring cuts after $n+1,\dots,n+p$ if r' cuts x. The relation between k and k' and the efficiency of r follow from the fact that r' solves either all of x, $n+1,\dots n+p$, or none at all, as proven in the claim below. The 'furthermore' part follows from Lemma 2, applied to r.

Claim. Either r' solves all breakpoints in $\{x, n+1,\dots,n+p\}$, or none at all.

Proof. For $p=2$, this is a direct application of Proposition 3 since elements x, $n+1$ and $n+2$ are in the same cycle of $\mathcal{C}(S',T')$.

For $p=3$, by Lemma 1, $C_x = \mathcal{C}_x(S',T')$ is a (k_x+1)-cycle containing x and $n+2$, and $\mathcal{C}(S',T')$ also contains a cycle denoted C_y with elements $n+1$ and $n+3$. By Proposition 3, r' solves any element in C_x (resp. C_y) iff it solves all elements in the same cycle (in particular, $k' \geq k_x+1$ if r' cuts x, so $k' = k+p$). Furthermore C_y is tied to C_x, so if r' solves C_y it must also solve C_x (by Proposition 4). It remains to check the last direction: if r' solves C_x, then it also solves C_y. Indeed, C_x is a k_x+1-cycle and r' solves a total of k_x+3 breakpoints, so it must also solve some 2-cycle C'_y. Aiming at a contradiction, assume that $C'_y \neq C_y$. Then C'_y is already a 2-cycle of $\mathcal{C}(S,T)$, and it is tied to some other cycle C'_x (both in $\mathcal{C}(S,T)$ and $\mathcal{C}(S',T')$), so r' also solves C'_x. Since C'_x may not be equal to C_x (x was chosen without ties), r' solves at least $|C_x| + |C'_y| + |C'_x| > k_x+3$ breakpoints, which yields a contradiction for a k_x+3-rearrangement.

□

Extending all Cycles. We use the natural order over integers as an arbitrary total order over the nodes. The *representative* of a cycle is its minimum node.

We assume (S,T) to be k-cyclic for some k. A *sample* for (S,T), where (S,T) is k-cyclic is a list X containing the representative from each k-cycle, and an arbitrary number of adjacencies. The p-extensions of (S,T) for sample $X = (x_1,\ldots,x_\ell)$ and of a rearrangement r of (S,T) are, respectively, $\Phi^p_X(S,T) = \phi^p_{x_\ell}(\ldots\phi^p_{x_2}(\phi^p_{x_1}(S,T))\ldots)$ and $\Psi^p_X(r) = \psi^p_{x_\ell}(\ldots\psi^p_{x_2}(\psi^p_{x_1}(r))\ldots)$.

For x_i in the sample, we write $n_{x_i} = |S| + p\cdot(i-1)$, i.e. n_{x_i} is the size of the strings on which $\phi^p_{x_i}$ is applied. Note that the above definition requires each x_i to be safe in $\phi^p_{x_{i-1}}(\ldots\phi^p_{x_1}(S,T)\ldots)$. This is indeed the case by Lemma 1: either $p=2$, there are no 2-cycles, and all breakpoints are without ties, or $p=3$, all 2-cycles are tied to a single cycle $\mathcal{C}_{x_j}(S,T)$ with $j<i$, which are all different from $\mathcal{C}_{x_i}(S,T)$ (since X is a sample and contains at most one element per cycle).

Proposition 5. *If (S,T) is k-cyclic, then $\Phi^2_X(S,T)$ is $(k+2)$-cyclic.*

Proof. This follows from Lemma 1, since the 2-extension adds 2 elements to each k-cycle, so $\Phi^2_X(S,T)$ is $(k+2)$-cyclic. □

Lemma 4. *If r is an efficient k-cut rearrangement of (S,T) then $r' = \Psi^p_X(r)$ is an efficient $k+p$ rearrangement of $(S',T') = \Phi^p_X(S,T)$. Moreover, in this case, $r(S,T)$ is k-cyclic with sample X, and $\Phi^p_X(r(S,T)) = r'(\Phi^p_X(S,T))$.*

Conversely, any efficient $k+p$ rearrangement r' of $(S',T') = \Phi^p_X(S,T)$ can be written as $r' = \Psi^p_X(r)$ where r is an efficient k-cut rearrangement of (S,T).

Proof. Given a k-cut rearrangement r, let

$$(S^0,T^0) = (S,T) \quad (S^j,T^j) = \phi^p_{x_j}(S^{j-1},Tj-1) \text{ for all } 0<j\le\ell$$
$$r_0 = r \qquad (r_j,r_j) = \psi^p_{x_j}(r_{j-1}) \text{ for all } 0<j\le\ell \textit{(Forward direction)}$$

Assuming that r is an efficient k-cut rearrangement of (S,T), since (S,T) is k-cyclic, by Proposition 3 r must solve a single cycle of $\mathcal{C}(S,T)$. Let x be the representative of this cycle: x is the only breakpoint of X cut by r, and $x = x_i$ for some i. Furthermore, $\mathcal{C}(r(S,T))$ is also k-cyclic with sample X (with one cycle less than $\mathcal{C}(S,T)$).

By Lemma 2 we have that r_j is an efficient k_j-cut rearrangement of (S^j,T^j) for each j, where $k_j = k_{j-1}$ if r_{j-1} does not cut x (i.e. $j\neq i$) and $k_j = k_{j-1}+p$ otherwise. So overall $r' = r_\ell$ is a an efficient $(k+p)$-cut rearrangement of $\Phi^p_X(S,T)$. The relationship $\Phi^p_X(r(S,T)) = r'(\Phi^p_X(S,T))$ also follows from Lemma 2.

The converse direction is proven similarly using Lemma 3, with a specific attention given to the size of the rearrangements: starting from r' (with $k+p$ cuts), the number of cuts remains constant, except for $\psi^p_{x_j}$ where it drops to k and then remains constant again (so the condition $k' \in \{k, k+p\}$ in Lemma 3 is indeed satisfied). □

Lemma 5. *If (S,T) is k-cyclic with sample X, then (S,T) is k-efficiently sortable if and only if $\Phi^p_X(S,T)$ is k-efficiently sortable.*

Proof. This is a direct application of Lemma 4: a sequence of efficient k-cut rearrangements of (S,T) translates into a sequence of efficient $(k+p)$-cut rearrangements of $\Phi_X(S,T)$ through function Ψ_X (note that X remains a sample of (S,T) throughout the sequence of rearrangements). □

Lemma 6. *For any odd $k \geq 5$, deciding whether a k-cyclic pair (S,T) is k-efficiently sortable is NP-hard. For any even $k \geq 6$, deciding whether a pair (S,T) is k-efficiently sortable is NP-hard.*

Proof. By induction on k. Deciding if a 3-cyclic pair (S,T) is efficiently sortable is NP-hard (cf. [3], where it is shown that deciding if a permutation can be sorted with $\frac{d_b(S,T)}{3}$ transpositions is NP-hard). For any $k \geq 5$, take $p=2$ if k is odd and $p=3$ otherwise, and consider a $(k-p)$-cyclic instance (S,T) and a sample X for (S,T) (note that one always exists): (S,T) is $(k-p)$-efficiently sortable iff $\Phi^p_X(S,T)$ is k-efficiently sortable by Lemma 5, and $\Phi^p_X(S,T)$ is k-cyclic for $p=2$ by Proposition 5. This gives a polynomial reduction proving hardness for k (even when restricted to k-cyclic permutations when k is odd). □

Theorem 7 is a corollary of Lemma 6, since a k-cyclic pair (S,T) is k-efficiently sortable iff S can be rearranged into T with no more than $\frac{d_b(S,T)}{k}$ k-cut rearrangements (Proposition 1).

Let Opt-SMCR be the optimisation version of SMCR, where we look for the smallest ℓ that is necessary to obtain T from S by k-cut rearrangements.

Theorem 8. Opt-SMCR *in permutations is 2-approximable.*

Proof. Let $I=(S,T,k)$ be an instance of Opt-SMCR. We first rewrite S and T into S' and T' in such a way that $T'=id_n$. Let $k'=\lfloor \frac{k}{2} \rfloor$. The algorithm consists in iterating the following three steps, starting from S': (a) rewrite S' by contracting adjacencies so as to obtain a permutation without fixed point, (b) cut around (i.e., right before and right after) the first k' elements $1,2,3\ldots k'$ of that permutation, and (c) rearrange it so as to obtain $id_{k'}$ followed by the rest of the permutation. Steps (b) and (c) above actually correspond to the case where k is even. If k is odd, (b) and (c) are slightly modified, since we are left with an unused cut: (b') do as (b) and additionally cut to the left of $k'+1$, (c') do as (c) but rearrange in such a way that k' and $k'+1$ are consecutive.

Clearly, the optimal value ℓ for Opt-SMCR satisfies $\ell \geq \frac{b(S,T)}{k}$. Our algorithm removes at least k' (at least $k'+1$) breakpoints at each iteration when k is even (when k is odd), and thus requires $\ell' \leq \frac{b(S,T)}{k'}$ ($\ell' \leq \frac{b(S,T)}{k'+1}$) many k-cut rearrangements. Altogether we have $\ell' \leq \frac{\ell k}{k'}$ if k is even and $\ell' \leq \frac{\ell k}{k'+1}$ if k is odd. Since $k'=\lfloor \frac{k}{2} \rfloor$, we conclude that $\ell' \leq 2\ell$. □

4 Conclusion

We introduced Sorting by Multi-cut Rearrangements, a generalization of usual genome rearrangement problems that do not incorporate reversals.

We discussed its classical computational complexity (P vs. NP-hard) and its membership in FPT with respect to the parameters ℓ and k. For this, we distinguished the case where S (and thus T) is a permutation from the case where it is a string.

The obvious remaining open problems are the ones indicated with a question mark in Tables 1 and 2, namely (a) the FPT status of SMCR with respect to parameter $\ell + k$ in strings, and (b) the computational complexity for constant ℓ and k part of the input in permutations. Extensions or variants of SMCR could also be studied, notably the one allowing reversals (and thus applicable to signed strings/permutations), or the one where T is the lexicographically ordered string derived from S. Finally, it would also be interesting to better understand the comparative roles of ℓ and k in SMCR, for instance by studying the following question: assuming k is increased by some constant c, what impact does it have on the optimal distance?

References

1. Bafna, V., Pevzner, P.A.: Genome rearrangements and sorting by reversals. SIAM J. Comput. **25**(2), 272–289 (1996)
2. Bafna, V., Pevzner, P.A.: Sorting by transpositions. SIAM J. Discret. Math. **11**(2), 224–240 (1998)
3. Bulteau, L., Fertin, G., Rusu, I.: Sorting by transpositions is difficult. SIAM J. Discrete Math. **26**(3), 1148–1180 (2012)
4. Bulteau, L., Komusiewicz, C.: Minimum common string partition parameterized by partition size is fixed-parameter tractable. In: Chekuri, C. (ed.) Proceedings of the Twenty-Fifth Annual ACM-SIAM Symposium on Discrete Algorithms, SODA 2014, Portland, Oregon, USA, January 5–7, 2014, pp. 102–121. SIAM (2014)
5. Christie, D.A.: Sorting permutations by block-interchanges. Inf. Process. Lett. **60**(4), 165–169 (1996)
6. Christie, D.A.: Genome rearrangement problems. Ph.D. thesis, University of Glasgow (1998)
7. Cygan, M., et al.: Parameterized Algorithms. Springer, Cham (2015). https://doi.org/10.1007/978-3-319-21275-3
8. Downey, R.G., Fellows, M.R.: Fundamentals of Parameterized Complexity. TCS. Springer, London (2013). https://doi.org/10.1007/978-1-4471-5559-1
9. Fertin, G., Labarre, A., Rusu, I., Tannier, E., Vialette, S.: Combinatorics of Genome Rearrangements. MIT Press, Computational molecular biology (2009)
10. Jansen, K., Kratsch, S., Marx, D., Schlotter, I.: Bin packing with fixed number of bins revisited. J. Comput. Syst. Sci. **79**(1), 39–49 (2013)
11. Goldstein, A., Kolman, P., Zheng, J.: Minimum common string partition problem: hardness and approximations. In: Fleischer, R., Trippen, G. (eds.) ISAAC 2004. LNCS, vol. 3341, pp. 484–495. Springer, Heidelberg (2004). https://doi.org/10.1007/978-3-540-30551-4_43
12. Pellestor, F., Gatinois, V.: Chromoanagenesis: a piece of the macroevolution scenario. Mol. Cytogenet. **13**(3) (2020). https://doi.org/10.1186/s13039-020-0470-0
13. Stephens, P.J., et al.: Massive genomic rearrangement acquired in a single catastrophic event during cancer development. Cell **144**, 27–40 (2011)