

Article

Sorting by Multi-Cut Rearrangements [†]

Laurent Bulteau ¹, Guillaume Fertin ^{2,*}, Géraldine Jean ² and Christian Komusiewicz ³

¹ LIGM, CNRS, Université Gustave Eiffel, F-77454 Marne-la-Vallée, France; laurent.bulteau@univ-eiffel.fr

² CNRS, LS2N, Université de Nantes, F-44000 Nantes, France; geraldine.jean@univ-nantes.fr

³ Fachbereich für Mathematik und Informatik, Philipps-Universität Marburg, D-35037 Marburg, Germany; komusiewicz@informatik.uni-marburg.de

* Correspondence: guillaume.fertin@univ-nantes.fr

[†] This paper is an extended version of our paper published in SOFSEM 2021.

Abstract: A *multi-cut rearrangement* of a string S is a string S' obtained from S by an operation called *k-cut rearrangement*, that consists of (1) cutting S at a given number k of places in S , making S the concatenated string $X_1 \cdot X_2 \cdot X_3 \cdot \dots \cdot X_k \cdot X_{k+1}$, where X_1 and X_{k+1} are possibly empty, and (2) rearranging the X_i s so as to obtain $S' = X_{\pi(1)} \cdot X_{\pi(2)} \cdot X_{\pi(3)} \cdot \dots \cdot X_{\pi(k+1)}$, 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 an alphabet 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 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 of massive simultaneous rearrangements. In this paper, we study the SMCR problem from an algorithmic complexity viewpoint. More precisely, we investigate its classical and parameterized complexity, as well as its approximability, in the general case or when S and T are permutations.



Citation: Bulteau, L.; Fertin, G.; Jean, G.; Komusiewicz, C. Sorting by Multi-Cut Rearrangements.

Algorithms **2021**, *14*, 169. <https://doi.org/10.3390/a14060169>

Academic Editor: Frank Werner

Received: 19 April 2021

Accepted: 27 May 2021

Published: 29 May 2021

Publisher's Note: MDPI stays neutral with regard to jurisdictional claims in published maps and institutional affiliations.



Copyright: © 2021 by the authors. Licensee MDPI, Basel, Switzerland. This article is an open access article distributed under the terms and conditions of the Creative Commons Attribution (CC BY) license (<https://creativecommons.org/licenses/by/4.0/>).

Keywords: genome rearrangements; sorting; strings; permutations; algorithmic complexity

1. Introduction

Genome rearrangements are large-scale evolutionary events that affect the genome of a species. They include among others reversals [1], transpositions [2], block interchanges [3] and many more [4]. Compared to small-scale evolutionary events such as the 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, called *chromoanagenesis*, has been shown to occur in genomes [5,6]. The term *chromoanagenesis* subsumes different types of rearrangements (namely, chromothripsis, chromoanasythesis 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 [6], 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 [5]. In this paper, we introduce a new model for genome rearrangements that is general enough to encompass most of the previously described genome rearrangements [4], as well as *chromoanagenesis*. In view of providing fast and accurate algorithms for simulating *chromoanagenesis*, and following a long tradition of algorithmic developments in the genome rearrangements community [4,7], our goal here is to study its properties in terms of computational complexity.

The present article is an extended and enriched version of a paper published by the same authors in SOFSEM 2021 [8].

1.1. Notations and Definitions

Given an alphabet Σ , let Σ^* be the set of all strings over Σ . We say that two strings $S, T \in \Sigma^*$ 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 with 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 special 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 there is no position $0 \leq j \leq n$ in T , such that the two consecutive characters S_i and S_{i+1} are consecutive in T , i.e., $T_j = S_i$ and $T_{j+1} = S_{i+1}$. For the special 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 .

Definition 1. Given a string $S \in \Sigma^*$ and an integer k , a k -cut rearrangement of S is an operation consisting of the two following steps: (1) cut S at k locations (thus S can be written as the concatenation of $k + 1$ strings, i.e., $S = X_1 \cdot X_2 \cdot X_3 \cdot \dots \cdot X_{k+1}$, where any 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_i s, so as to obtain $S' = X_{\pi(1)} \cdot X_{\pi(2)} \cdot X_{\pi(3)} \cdot \dots \cdot X_{\pi(k+1)}$, 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_i s considered in a given k -cut rearrangement will be called a block.

In the above model, any block is only allowed to move whenever it is cut at both its left and right extremities. Hence, moving a prefix of S (resp. a suffix of S) comes at the cost of an “extra cut” to its left (resp. right), and this corresponds, in the above definition, to the case where X_1 (resp. X_{k+1}) is empty.

Note also that, although a k -cut rearrangement has been defined as a cut along S at k locations, it is always possible, if necessary, to perform only $k' \leq k$ cuts, by cutting several times at the same location. Thus, one may mimic a k -cut rearrangement, while actually realizing a k' -cut rearrangement. Note finally that, in this model, each block X_i can only be moved, thus no reversal of 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 k and ℓ .

Question: Is there a sequence of at most ℓ many k -cut rearrangements that transforms S into T ?

Some examples of k -cut rearrangement scenarii are presented in Figure 1. For convenience, we may also refer to the SMCR problem with parameters k and ℓ as the (k, ℓ) -SMCR problem. Note that this notation shall not imply that k or ℓ are constants. Unless mentioned otherwise, input strings are considered to have an unbounded alphabet, although most hardness proofs also hold for constant-size alphabets, as detailed in the tables of results.

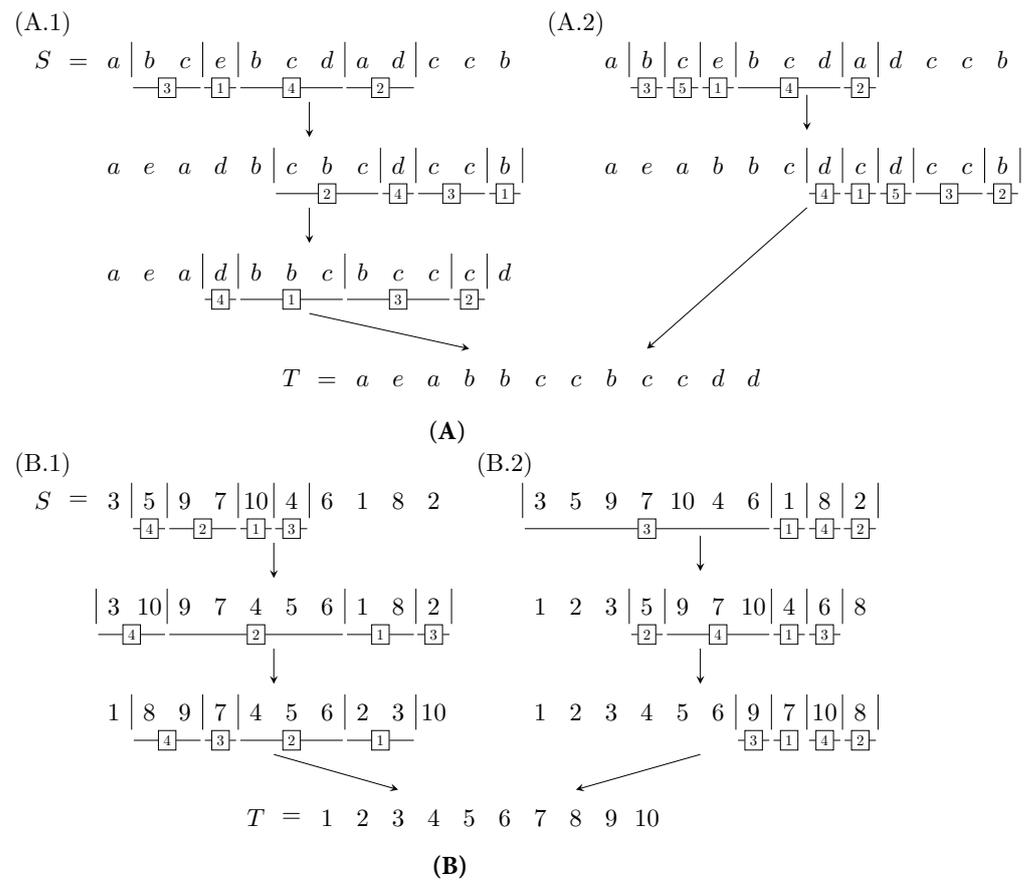


Figure 1. Examples of k -cut rearrangement scenarios. Cuts are represented by vertical lines and the order of the rearranged blocks is indicated by the framed numbers. **(A)** Example on strings $S = abcbebcdadccb$ and $T = aeabbccbccdd$. **(A.1)** shows a 5-cut rearrangement scenario with $\ell = 3$. **(A.2)** shows a 6-cut rearrangement scenario with $\ell = 2$. **(B)** Example on permutations $S = 35971046182$ and $T = 12345678910$ with $b(S, T) = 11$. **(B.1, B.2)** show two alternative 5-cut rearrangement scenarios with $\ell = 3$.

1.2. Parameterized Algorithmics

Our goal in this paper is to study the algorithmic properties of SMCR. In particular, we aim to outline the effect of bounds on k and ℓ on the complexity of the problem. To carry out our studies, we use the framework of parameterized algorithmics [9,10] which has been applied to many hard string problems in the past [11].

An instance of a parameterized problem consists of the (classical) input instance I and a parameter $\kappa \in \mathbb{N}$. A parameterized problem is *fixed-parameter tractable* if every instance (I, κ) can be solved in $f(\kappa) \cdot \text{poly}(n)$ time, where $n = |I|$ is the size of the input, and f is some computable function. An algorithm with such a running time is called an FPT-algorithm. Thus, an FPT-algorithm can be considered practical if f does not grow too fast and κ takes on relatively small values, for example if $f(\kappa) = 2^\kappa$ and κ is upper-bounded by $\log |I|$.

A particularly important technique for obtaining FPT-algorithms is *kernelization*, which is a framework for analyzing the power of data reduction rules for parameterized problems. A parameterized problem admits a kernelization, or simply a *kernel*, if there exists a polynomial-time algorithm that transforms every instance (I, κ) into an equivalent instance (I', κ') , such that $|I'| + \kappa' \leq g(\kappa)$. The function g is called the *kernel size*. For any decidable parameterized problem L , the existence of a kernel implies that L has an FPT-algorithm with respect to κ , since the instance (I', κ') can be solved in $f(\kappa)$ time.

In our study, we consider the case that the parameter κ is one of k or ℓ , that κ is the combined parameter $k + \ell$, or that k and ℓ are part of the input. Moreover, we consider some special cases where k or ℓ take on small constant values. Some of these cases correspond to well-studied problems, such as computing the transposition distance or the MCSP distance, as we discuss below. We will separately consider the case where S and T are strings (i.e., $d > 1$ both in S and T) and the case where S and T are permutations (i.e., $d = 1$ both in S and T), in Sections 2 and 3, respectively.

1.3. Basic Observations

Both in permutations and strings, the cases $k = 1$ and $k = 2$ are trivial, since they do not allow one to move any block, and thus we are in the presence of a YES-instance if, and only if, $S = T$. Additionally, the SMCR problem is a natural generalization and extension of several 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 (see Figure 2). Thus SMCR in that case is equivalent to the SORTING BY TRANSPOSITIONS problem [2], which is NP-hard, even if S and T are permutations [12] or binary strings [13].

When $k = 4$, each k -cut rearrangement allows one to move two blocks among X_2, X_3 and X_4 , which exactly corresponds to the SORTING BY BLOCK INTERCHANGE problem (see Figure 2). This problem is known to be in P for permutations [3] and NP-hard for strings (an NP-hardness proof for binary strings is given in ([14], Theorem 5.7.2)).

It may come as a surprise that the case $k = 4$ is polynomial on permutations, when the case $k = 3$ —and all other non-trivial cases—are NP-hard. The original algorithm for $k = 4$ is based on the cycle graph (see Section 3.1.1 for formal definitions): in this setting, the goal is to create as many cycles as possible at each step (a block interchange may create at most two cycles). The key intuition is that such an “efficient” block-interchange always exists and can be polynomially computed. In other words, efficient block-interchanges commute, so it is safe to greedily apply the best-available operation at any point. This is in contrast with $k = 3$, since transpositions may also create up to two new cycles, but such efficient transpositions are not always guaranteed to exist and cannot be chosen greedily (applying one may destroy another). In fact, finding a sequence of such efficient transpositions is at the core of the NP-hardness proof [12]. On a related note, Lin et al. [15] proposed an algebraic formulation of the SORTING BY BLOCK INTERCHANGE problem, translating the cycles into permutation orbits, leading to a linear-time algorithm.

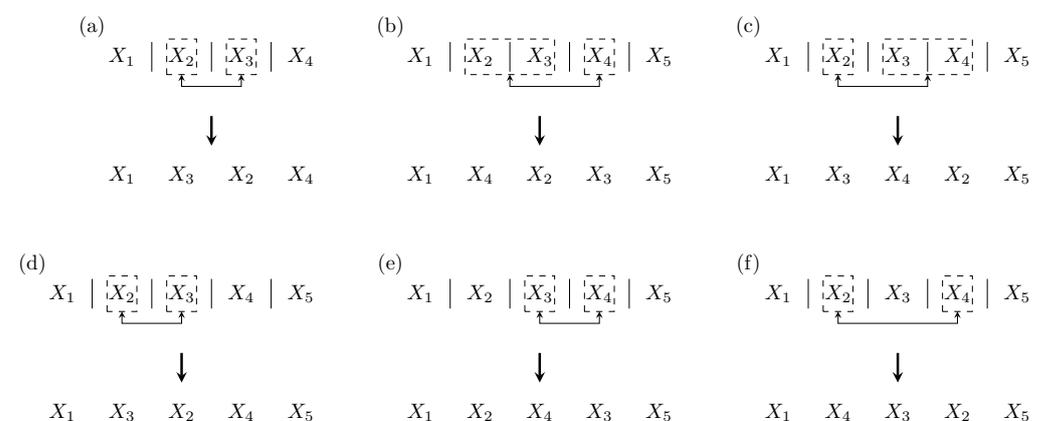


Figure 2. Relationship between classical rearrangements and k -cut rearrangements. Cuts are represented by vertical lines, while dashed rectangles define blocks that are moved in classical rearrangements. (a) when $k = 3$, a k -cut rearrangement corresponds to a transposition (here of blocks X_2 and X_3). (b–f) are the 5 possible cases that we can get with a 4-cut rearrangement, all of them corresponding to a block interchange of dashed blocks.

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 if and only if $b(S, T) \leq k$. In strings, the SMCR problem resembles the MINIMUM COMMON STRING PARTITION problem [16]. Both problems are indeed almost identical, and only differ in their optimum value, depending on whether S and T share the same prefix and/or suffix. This will be discussed in more detail in Theorems 5 and 6.

When k and ℓ 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 permitted moves, has a running time of $O(n^{k\ell+1})$ —the additional factor of n 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 $S = \text{afedcbg}$, $T = \text{abcdefg}$, $k = 3$, and $\ell = 2$. This is a YES-instance for $(6, 1)$ -SMCR: One 6-cut rearrangement on S with the cuts a|f|e|d|c|b|g (symbolized as vertical segments) suffices to obtain T . In contrast, it is a NO-instance for $(3, 2)$ -SMCR: The number $b(S, T)$ of breakpoints is equal to 6 and every 3-cut rearrangement is a transposition. In this instance, however, 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. Recall that d denotes the maximum number of occurrences of any character in the input string S .

$\ell \backslash k$	$O(1)$	parameter	part of the input
1	P	FPT (Theorem 6)	NP-hard: for $\ell = 1$ even when $d = 2$ (Theorem 5) for any fixed $\ell \geq 1$ (Theorem 4)
$O(1)$	(Section 1.3)		
parameter	open		
part of the input	for any $k \geq 5$ even in k -ary strings (Theorem 1) for any odd $k \geq 7$ even in ternary strings (Theorem 3) for $k = 3, 4$ even in binary strings [13,14]		

As mentioned in the previous section, we know that SMCR is NP-hard in binary strings for $k = 3, 4$ [13,14]. In the following theorem, we extend this result to any value of k , however, in larger alphabet strings.

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

Proof. By reduction from 3-PARTITION, a 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 . Observe that 3-PARTITION is strongly NP-hard, that is, it is NP-hard even if all integers are represented as unary numbers [17]. Given an instance of 3-PARTITION (\mathcal{A}, B) with $\mathcal{A} = \{a_1, a_2, \dots, a_{3m}\}$ and $mB = \sum_{i=1}^{3m} a_i$, we construct an instance of SMCR for any fixed $k \geq 5$ as follows. By simple padding arguments, we may assume that each a_i is a multiple of $4m$ and that $\frac{B}{4} < a_i < \frac{B}{2}$. This implies the following property: If for some subset I of $\{1, 2, \dots, 3m\}$ and some δ with $0 \leq \delta \leq 4m$ 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 $k - 1 \cdot k - 2 \cdot \dots \cdot 3$, and by X' the reverse of X , i.e., $3 \cdot 4 \cdot \dots \cdot k - 1$. Note that X and X' have length $k - 3 \geq 2$. We define

$$S = 10^{a_1} 10^{a_2} \dots 10^{a_{3m}} 1(20X0X0X0)^m 2$$

and

$$T = (1X')^{3m} 1(20^{B+4})^m 2,$$

and set $\ell = 3m$. This completes the construction. Before proving its correctness, let us group length-2 substrings of S and T (hereafter called *duos*) based on whether they are in excess in S , in T , or equal in both strings.

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

There are no further duos in S or T . To show the correctness of the reduction, we show that (\mathcal{A}, B) is a YES-instance of 3-PARTITION if, and only if, there exists a sequence of at most $\ell = 3m$ 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 $3m$ many k -cut rearrangements described above is T .

(\Leftarrow) There are altogether $6m + (k - 2)3m = 3km$ duos in Group 1 which are in excess in S , and $3km$ duos in Group 2 which are in excess in T . Since $\ell = 3m$, each k -cut rearrangement cuts k duos in Group 1 (and no duo in Group 2 or 3). In particular, no 00 duo 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 $4m$ of these singleton 0s, we have $0 \leq \delta \leq 4m$. 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 . \square

Theorem 1 is improved by the following two theorems, which prove the NP-hardness of SMCR in ternary strings, but limited to odd values of k with $k \geq 7$. More precisely, Theorem 2 focuses on the specific case $k = 7$, while Theorem 3 extends the result to any odd $k \geq 7$.

Theorem 2. *SMCR is NP-hard for $k = 7$ in ternary strings.*

Proof. The proof is, as the proof of Theorem 1, by reduction from the strongly 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 triplets, such that the sum of the integers in each triplet is equal to B . Given an instance (\mathcal{A}, B) of 3-PARTITION with $\mathcal{A} = \{a_1, a_2, \dots, a_{3m}\}$ such that (i) $\frac{B}{4} < a_i < \frac{B}{2}$ for any $1 \leq i \leq 3m$ and (ii) $mB = \sum_{i=1}^{3m} a_i$, we build the following instance (S, T, k, ℓ) of SMCR:

$$S = 10^{a_1} 10^{a_2} \dots 10^{a_{3m}} 12^{m+1}$$

$$T = 1^{3m+1} (20^B)^m 2$$

with $k = 7$ and $\ell = m$. Clearly, S and T are built on an alphabet of cardinality 3. Now, let us prove the correctness of our reduction.

(\Rightarrow) Suppose the instance (\mathcal{A}, B) of 3-PARTITION is a YES-instance. Thus, \mathcal{A} can be partitioned in sets A_1, A_2, \dots, A_m , such that $A_j = \{a_{j_1}, a_{j_2}, a_{j_3}\}$ and $a_{j_1} + a_{j_2} + a_{j_3} = B$ for any $1 \leq j \leq m$. We now show how to reach T from S , using 7-cut rearrangements, in $\ell = m$ steps: at each step $1 \leq j \leq m$, cut in S to the right of the 1 preceding $0^{a_{j_1}}$ (resp. $0^{a_{j_2}}, 0^{a_{j_3}}$) and to the left of the 1 following $0^{a_{j_1}}$ (resp. $0^{a_{j_2}}, 0^{a_{j_3}}$), for a total of 6 cuts; the last cut is to the right of the j -th occurrence of 2. The 6 first cuts allow one to move blocks $0^{a_{j_1}}, 0^{a_{j_2}}, 0^{a_{j_3}}$, which we concatenate so as to obtain 0^B (since $a_{j_1} + a_{j_2} + a_{j_3} = B$ by hypothesis). The 7th cut allows us to insert block 0^B between the j -th and the $(j + 1)$ -th copy of 2. An illustration of the first step of the above described algorithm for solving SMCR is provided in Figure 3.

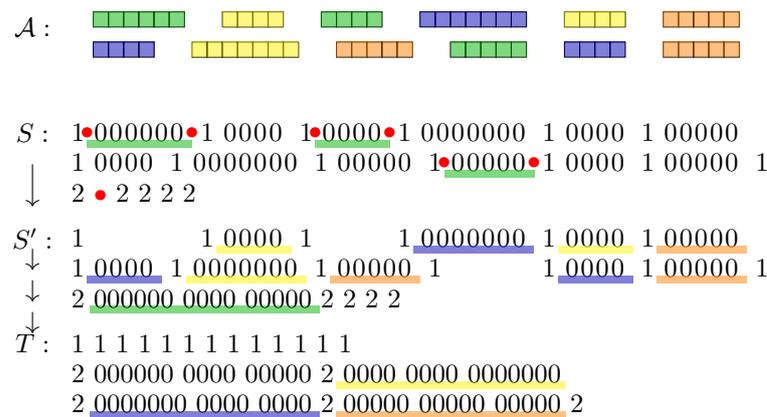


Figure 3. Example of the proposed reduction starting from the following instance of 3-PARTITION: $\mathcal{A} = \{6, 4, 4, 7, 4, 5, 4, 7, 5, 5, 4, 5\}$ (thus $m = 4$) and $B = 15$. Each a_i is represented by a rectangle containing a_i squares. Colors in \mathcal{A} describe a way to partition \mathcal{A} into triplets, each summing to $B = 15$. S and T are built as described in the reduction. Each vertical arrow corresponds to a 7-cut. The first one, which is the first step of the proposed 7-cut rearrangement scenario, going from S to S' , is detailed: each red bullet corresponds to a cut, and blocks of 0 that are rearranged are underlined in green (corresponding to the three “green a_i s” summing to B).

After every step $1 \leq j \leq m$ is achieved, three of the 0^{a_i} blocks (each one being surrounded by 1s) are removed, while a 0^B block is created between two occurrences of 2. Thus, after step m , we have produced sequence $1^{3m+1}(20^B)^m 2 = T$. Since every step uses $k = 7$ cuts and T is obtained after $m = \ell$ steps, we conclude that (S, T, k, ℓ) is a YES-instance for SMCR.

(\Leftarrow) Suppose that the instance (S, T, k, ℓ) that we have built is a YES-instance. First, we note that S contains exactly $7m$ duos (i.e., length-2 substrings) that are absent from T , while T contains $7m$ duos that are absent from S , as shown in the exhaustive list of duos provided in Table 2.

Table 2. Exhaustive list of duos present both in S and T , together with their occurrences.

Duo	Occurrences in S	Occurrences in T
(0, 0)	$Bm - 3m$	$Bm - m$
(0, 1)	$3m$	0
(0, 2)	0	m
(1, 0)	$3m$	0
(1, 1)	0	$3m$
(1, 2)	1	1
(2, 0)	0	m
(2, 1)	0	0
(2, 2)	m	0

Thus, starting from S , and in order to end in T , there are altogether $7m$ duos to destroy, and $7m$ to create. Since $k = 7$ and $\ell = m$, for each of the m allowed steps, 7 duos need to be destroyed (and never further created), while 7 duos need to be created (and never further destroyed). In particular, this implies that we never cut strictly inside a block of the form 0^{a_i} , otherwise a (0, 0) duo would be destroyed (whereas the goal is to create altogether $2m = (B - 1)m - (B - 3)m$ of them). Since, in T , blocks containing 0s all are of the form 0^B , this implies that there is a way to arrange all blocks of the form 0^{a_i} from S , by concatenating them (and never cutting them), so that we obtain m blocks of the form 0^B . This shows that it is possible to partition \mathcal{A} into m sets, each achieving a sum equal to B . However, by hypothesis, each a_i satisfies $\frac{B}{4} < a_i < \frac{B}{2}$. Hence, the only way to partition \mathcal{A} in the above described manner is to group the a_i s by triplets. Consequently, this shows that (\mathcal{A}, B) is a YES-instance for 3-PARTITION. \square

Proof of Theorem 2 above can be extended so as to prove NP-hardness of SMCR for any odd $k \geq 7$ in ternary alphabets. This is the purpose of the following theorem.

Theorem 3. SMCR is NP-hard for any odd $k \geq 7$ in ternary strings.

The proof is heavily based on the arguments developed in proof of Theorem 2. The idea here is to reduce from a decision problem that we call p -PARTITION ($p \geq 3$ being an integer), which is an extension of 3-PARTITION; second, we slightly modify strings S and T according to this new problem. First, let us formally introduce p -PARTITION, and discuss its NP-hardness. The input of p -PARTITION is a set $\mathcal{A}' = \{a'_1, a'_2, \dots, a'_{pm}\}$ and an integer B' , such that $mB' = \sum_{i=1}^{mp} a'_i$, and the question is whether \mathcal{A}' can be partitioned into sets A'_1, A'_2, \dots, A'_m , each of cardinality p , and such that, for any $1 \leq j \leq m$, $\sum_{a'_i \in A'_j} a'_i = B'$. We call BALANCED p -PARTITION the restriction where $a'_i > \frac{B'}{p+1}$ for all i .

Lemma 1. For any $p \geq 3$, BALANCED p -PARTITION is strongly NP-hard.

Proof. First, we show that for any $p \geq 3$, p -PARTITION is strongly NP-hard, by reduction from 3-PARTITION. Indeed, consider an instance (\mathcal{A}, B) of 3-PARTITION, with $\mathcal{A} = \{a_1, a_2, \dots, a_{3m}\}$ and $B = \frac{1}{m} \sum_{i=1}^{3m} a_i$. Let $N = B + 1$. We now construct an instance (\mathcal{A}', B') of p -PARTITION as follows: for any $1 \leq i \leq 3m$, $a'_i = a_i$, while $a'_i = N$ for all $3m + 1 \leq i \leq pm$. Let $B' = \frac{1}{m} \sum_{i=1}^{pm} a'_i$. Then, $B' = \frac{1}{m} (\sum_{i=1}^m a_i + (p - 3)mN)$, that is $B' = B + (p - 3)N$. Note that $B' < (p - 2)N$.

(\Rightarrow) If (\mathcal{A}, B) is a YES-instance for 3-PARTITION, then assume that \mathcal{A} can be partitioned into A_1, A_2, \dots, A_m . Now, for any $1 \leq j \leq m$, take A'_j to be the union of A_j and of any $(p - 3)$ of the remaining elements among $a'_{3m+1}, a'_{3m+2}, \dots, a'_{pm}$. The sum of the elements

in A'_j is equal to B' , and sets A'_j together form a partition of \mathcal{A}' . Hence, (\mathcal{A}', B') is a YES-instance for p -PARTITION.

(\Leftarrow) If (\mathcal{A}', B') is a YES-instance for p -PARTITION, then assume that \mathcal{A}' can be partitioned into A'_1, A'_2, \dots, A'_m . Any A'_j contains at least three elements among $\{a'_1, a'_2, \dots, a'_{3m}\}$, otherwise the sum of elements of A'_j would be at least $(p - 2)N > B'$. Since there are $3m$ such elements, we conclude that each A'_j must contain exactly three elements of \mathcal{A} . Now, let $A_j = A'_j \cap \mathcal{A}$. Let S_j (resp. S'_j) be the sum of the elements of A_j (resp. A'_j). Then, we have $S'_j = S_j + (p - 3)N$. Since $S'_j = B'$, we conclude that $S_j = B$. This shows that A_1, A_2, \dots, A_m is a partition of \mathcal{A} for which each set is of cardinality 3 and sums to B . In other words, (\mathcal{A}, B) is a YES-instance for 3-PARTITION.

Finally, the following argument shows that p -PARTITION remains strongly NP-hard in the balanced restriction. Indeed, take any instance of p -PARTITION and modify it as follows: every a'_i is increased by B' , and set $B'' = (p + 1)B'$. In that case, the sum of all elements from \mathcal{A}' is $m(p + 1)B'$, and if a partition A'_1, A'_2, \dots, A'_m exists, the sum of the elements of each A'_j is equal to B'' . Since $a'_i > B'$, we conclude that $a'_i > \frac{B''}{p+1}$ and we are done. Note that each a'_i remains bounded by a polynomial on the instance size (since each a_i and B is polynomially bounded in 3-PARTITION), hence preserving the strong NP-hardness. \square

Proof of Theorem 3. Now, in order to prove that SMCR is NP-hard in ternary strings for any odd $k \geq 7$, it suffices to adapt the reduction provided in proof of Theorem 2.

Hence, starting from any instance (\mathcal{A}', B') of BALANCED p -PARTITION, we build the following instance (S, T, k, ℓ) of SMCR:

$$S = 10^{a'_1} 10^{a'_2} \dots 10^{a'_{pm}} 12^{m+1}$$

$$T = 1^{pm+1} (20^{B'})^m 2$$

with $k = 2p + 1$ and $\ell = m$. Clearly, S and T are built on an alphabet of cardinality 3.

Correctness of the reduction relies on the same type of arguments, as in proof of Theorem 2, and are only informally described here: if a partition of \mathcal{A}' exists, this provides us with a rearrangement scenario where, at each step, $2p + 1$ cuts are applied. More precisely, $2p$ of these cuts allow one to “disconnect” the p blocks of the form $0^{a'_i}$ corresponding to a set A'_j (of cardinality p) of the partition, while the last cut allows one to insert the concatenated abovementioned blocks into a single $0^{B'}$ block between two occurrences of character 2. The reverse direction is based on the number of duos that need to be deleted and created: there are $(2p + 1)m$ duos in S which are not present in T , and $(2p + 1)m$ duos in T which are not present in S ; besides, $(2p + 1)m = k\ell$ is the total number of cuts we are allowed altogether in a rearrangement scenario from S to T . Consequently, no $(0, 0)$ duo can be deleted during such a scenario, which shows that any block of the form $0^{B'}$ in T must be the concatenation of blocks of the form $0^{a'_i}$. This in turn proves that \mathcal{A}' can be partitioned in sets, each of which sums to B' . Moreover, since every $a'_i > \frac{B'}{p+1}$, all sets in that partition are of cardinality at most p . However, since there are m sets and altogether mp elements to partition, we conclude that each set is necessarily of cardinality exactly p . Consequently, we are in the presence of a YES-instance for p -PARTITION. \square

Theorem 3 above shows that SMCR is NP-hard for ternary strings, for any odd $k \geq 7$. We conjecture that this result can be extended in two ways, i.e., that SMCR is NP-hard for any $k \geq 5$ (both odd and even), even in the case of binary strings.

Theorems 1 to 3 are concerned with the complexity of SMCR relatively to parameter k . We now turn our attention to parameter ℓ .

Theorem 4. SMCR is NP-hard for $|\Sigma| = 5$ and any fixed $\ell \geq 1$.

Proof. The reduction being very similar to the one of Theorem 1, we only highlight here the differences that allow us 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 in proof of Theorem 1 using $k' = 5$ (i.e., 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 Section 1.3. 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 as in proof of Theorem 1 holds, namely $3k'm = 15m$ duos need to be broken using ℓ many k -cut rearrangements, with $\ell k = 15m$. So, again, no $(0, 0)$ duo may be broken, and by the same argument, we obtain a valid 3-partition of \mathcal{A} . \square

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

Proof. The proof is obtained by reduction from the MINIMUM COMMON STRING PARTITION (MCSP) problem, which has been proven to be NP-hard in strings, even when $d = 2$ [16]. 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 $S = X_1 \cdot X_2 \cdot \dots \cdot X_{p-1} \cdot X_p$ and T can be written as $T = X_{\pi(1)} \cdot X_{\pi(2)} \cdot \dots \cdot X_{\pi(p-1)} \cdot X_{\pi(p)}$, 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 SMCR by 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 for solving SMCR. Conversely, if (S', T', k, ℓ) is a YES-instance for SMCR, and since x occurs only once in S' , then 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. \square

Note that MCSP has been proved to be in FPT with respect to the size of the solution [18]. It can be seen that this result can be adapted for the SMCR problem in the case $\ell = 1$.

Theorem 6. *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 $S_{a,b}$ and $T_{a,b}$ be the strings obtained from S and 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 [18]. \square

Note that it is not sufficient to check only with the longest common prefix and suffix (i.e., $S_{A,B}$ and $T_{A,B}$), which can be seen in the following example. When $S = \text{xxzyxtyy}$ and $T = \text{xxtyxzzy}$, then S can be transformed into T via the 3-cut rearrangement with the cuts $x|zxy|xty|y$. Moreover, we have $A = B = 2$, but only $S_{1,1} = \text{zyxty}$ and $T_{1,1} = \text{xtyzzy}$

have a common partition into two blocks, whereas $S_{1,1} = zyxt$ and $T_{2,2} = tyxz$ do not have such a partition.

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 3.

Table 3. Summary of the results for SORTING BY MULTI-CUT REARRANGEMENTS in permutations. Additionally, OPT-SMCR admits a 2-approximation algorithm (Theorem 10).

$\ell \backslash k$	3	4	$O(1)$	parameter	part of the input
1	P (Section 1.3)				
$O(1)$					
parameter	FPT (Theorem 9)			FPT (Theorem 9)	NP-hard: For $\ell \geq 2$ (Theorem 8)
part of the input	NP-hard [12]				For $k \geq 5$ (Theorem 7)

We start with showing the NP-hardness for constant k or ℓ (Theorems 7 and 8). Then, we show fixed-parameter tractability for the combined parameter $k + \ell$ in Theorem 9. Finally, we provide a 2-approximation result (Theorem 10).

3.1. Hardness for Constant Number of Cuts

We now show that SMCR is also hard in permutations.

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

Theorem 7 gives a complexity dichotomy with respect to the number of cuts in the rearrangement, since the case $k = 3$ is known to be NP-hard [12] and the case $k = 4$ is known to be polynomial-time solvable [3].

We show Theorem 7 by reduction from SORTING BY TRANSPOSITIONS on 3-cyclic permutations [12]. Intuitively, in such permutations, it is straightforward to identify triples of breakpoints, called 3-cycles, that should be solved together in a transposition. The difficulty arises in selecting a correct order in which those 3-cycles should be solved. Our approach consists of 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 $k \geq 5$.

3.1.1. Breakpoints and Cycle Graph

See Figure 4 for an illustration of the definitions in this section. For a permutation S of length n , we assume that the alphabet of S is $\{1, 2, \dots, n\}$. We further write $S_0 = 0$ and $S_{n+1} = n + 1$ (they are considered as *implicit* elements: they may not be part of rearrangements, but allow all other characters to have another element before and after). 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 $\{0, 1, \dots, n\}$ with arcs $T_j \rightarrow S_i$ if $T_{j+1} = S_{i+1}$. Every vertex has indegree and outdegree 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 $x \rightarrow y$ in $\mathcal{C}(S, T)$ is y (or equivalently, the outgoing arc of y).

We write $C_x(S, T)$ for the cycle of $C(S, T)$ containing element x . A cycle graph (and, by extension, a pair of sequences generating this cycle graph) is k -cyclic 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 r cuts between $x = S_i$ and S_{i+1} . Furthermore, r solves breakpoint x if r cuts x and x is an adjacency in $r(S)$. A rearrangement r solves a cycle if it solves all breakpoints in it. We write $d_b(S, T)$ for the number of breakpoints of $C(S, T)$. 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.

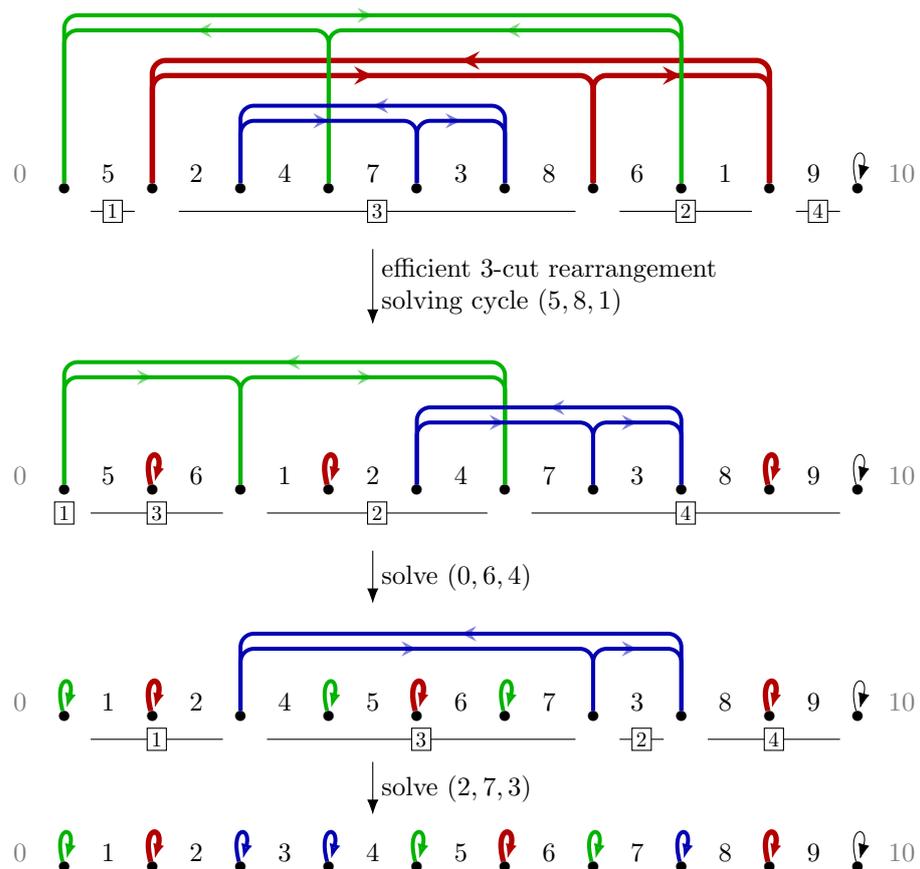


Figure 4. Example of a 3-cut rearrangement scenario from $S = 524738619$ to $T = 123456789$ using cycle graphs. From top to bottom: The cycle graph for (S, T) contains three non-trivial cycles and a trivial one (breakpoint u is depicted to the right of the label for u). Blocks 1 to 4 of a 3-cut rearrangement (cutting after 5, 8, and 1) are given: it solves cycle $5 \rightarrow 8 \rightarrow 1 \rightarrow 5$ (denoted $(5, 8, 1)$ for short). Two other efficient rearrangements solve cycles $(0, 6, 4)$ and $(2, 7, 3)$, thus reaching T in three steps. Note that some cycles do not admit efficient 3-cut rearrangement, e.g., $(0, 6, 4)$ in the original permutation S .

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 $x \rightarrow y$ 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 $C(S, T)$. \square

Proposition 3. *If r is efficient, it solves a cycle if, and only if, 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). Due to 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. \square

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 $T_{j+1} = y$ for some j (for an illustration, see Figure 5). A breakpoint is *without ties* if no cycle is tied to the cycle containing it.

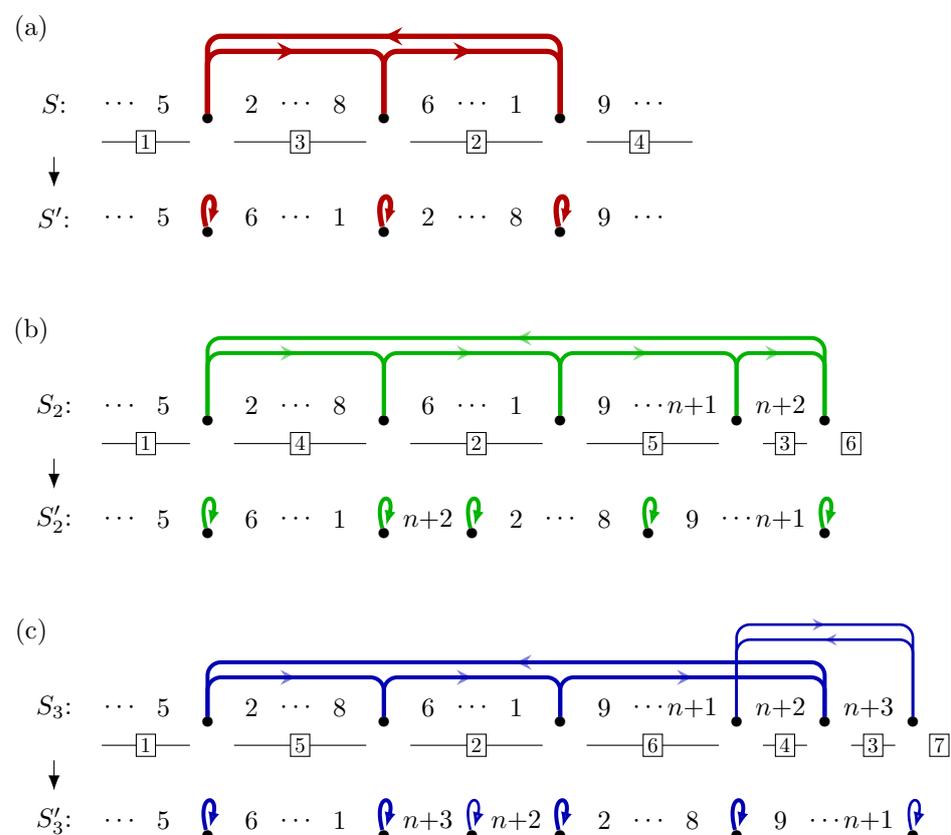


Figure 5. (a) The cycle (1, 5, 8) from Figure 4, and how solving it affects the rest of the sequence (substrings 2 – 8 and 6 – 1, i.e., blocks $\boxed{3}$ and $\boxed{2}$ are swapped). (b) The 2-extension of S for breakpoint $x = 1$, giving cycle (1, $n + 1$, $n + 2$, 5, 8). This extended cycle can be solved with a 5-cut rearrangement having an equivalent effect on the sequence (substrings 2 – 8 and 6 – 1 are swapped). The resulting S'_2 is the 2-extension of S' for adjacency $x = 1$. (c) The 3-extension of S for breakpoint $x = 1$, giving cycles (1, $n + 2$, 5, 8) and ($n + 1$, $n + 3$) (the latter being tied to the former: there is no way of solving breakpoint $n + 3$ without cutting $n + 2$). These two cycles can be solved with a 6-cut rearrangement having an equivalent effect on the sequence (again, substrings 2 – 8 and 6 – 1 are swapped). The resulting S'_3 is the 3-extension of S' for adjacency $x = 1$.

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 . \square

3.1.2. One-Cycle Extensions

We now introduce the central technical engine for the reduction of Theorem 7. 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 (see Figure 5 for an example):

- For $p = 2$:

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

$$\begin{aligned} S' &= (S_1, \dots, S_i = x, n + 3, n + 2, S_{i+1}, \dots, S_n, n + 1) && \text{if } x \text{ is an adjacency} \\ S' &= (S_1, \dots, S_n, n + 1, n + 2, n + 3) && \text{if } x \text{ is a breakpoint} \\ T' &= (T_1, \dots, T_j = x, n + 3, n + 2, T_{j+1} = S_{i+1}, \dots, T_n, n + 1) \end{aligned}$$

Note that the implicit rightmost element $S_{n+1} = n + 1$ is always inserted explicitly (indeed, S_n is followed by $n + 1$ in both S' and T'), and becomes $S'_{n+p+1} = T'_{n+p+1} = n + p + 1$ in the new permutations.

Lemma 2. *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 $T_j \rightarrow S_i$ with $T_j \neq x$ in $\mathcal{C}(S, T)$. Since no element is inserted after T_j or S_i , $T_j \rightarrow S_i$ 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 $T_j \rightarrow S_j$ 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 duos 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 $\{x, n + 1, \dots, n + p\}$. 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 $x \rightarrow S'_{n+p-1} = n + p - 1$ in $\mathcal{C}(S', T')$ (note, in particular, that y no longer has its incoming arc $x \rightarrow y$).

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 $n + 1 \rightarrow S'_{n+p} = n + p$ 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 $n + 3 \rightarrow S'_{n+1} = n + 1$ in $\mathcal{C}(S', T')$.

At this point, the outgoing arcs for all vertices except $n + 2$ have been described, as well as incoming arcs for all vertices except y , so the last remaining arc is $n + 2 \rightarrow y$.

Overall, for $p = 2$, arc $x \rightarrow y$ is replaced with the path $x \rightarrow n + 1 \rightarrow n + 2 \rightarrow y$. For $p = 3$, arc $x \rightarrow y$ is replaced with $x \rightarrow n + 2 \rightarrow y$ 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 $(n + 2, n + 3)$. \square

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 $\phi_x^p(S, T)$ are related through ψ_x^p .

Lemma 3. *If r is an efficient k -cut rearrangement of (S, T) , then $r' = \psi_x^p(r)$ is an efficient k' -cut 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 cases, 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). \square

The following claim will be useful for proving Lemma 4.

Claim 1. *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 2, $C_x = \mathcal{C}_x(S', T')$ 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) if, and only if, 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). Now, all that is left is 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 $(k_x + 3)$ -cut rearrangement. \square

Lemma 4. *If r' is an efficient k' -cut 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 $r' = \psi_x^p(r)$, 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 3: 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 Claim 1. The 'furthermore' part follows from Lemma 3, applied to r . \square

3.1.3. 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 $X = (x_1, \dots, x_\ell)$ and of a rearrangement r of (S, T) are, respectively, $\Phi_X^p(S, T) = \phi_{x_\ell}^p(\dots \phi_{x_2}^p(\phi_{x_1}^p(S, T)) \dots)$ and $\Psi_X^p(r) = \psi_{x_\ell}^p(\dots \psi_{x_2}^p(\psi_{x_1}^p(r)) \dots)$.

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 $\phi_{x_i}^p$ is applied. Note that the above definition requires each x_i to be safe in $\phi_{x_{i-1}}^p(\dots \phi_{x_1}^p(S, T) \dots)$. This is indeed the case by Lemma 2: 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 $\Phi_X^2(S, T)$ is $(k + 2)$ -cyclic.*

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

Lemma 5. *If r is an efficient k -cut rearrangement of (S, T) , then $r' = \Psi_X^p(r)$ is an efficient $k + p$ rearrangement of $(S', T') = \Phi_X^p(S, T)$. Moreover, in this case, $r(S, T)$ is k -cyclic with sample X , and $\Phi_X^p(r(S, T)) = r'(\Phi_X^p(S, T))$.*

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

Proof. Given an efficient k -cut rearrangement r , let

$$\begin{aligned} (S^0, T^0) &= (S, T), & (S^j, T^j) &= \phi_{x_j}^p(S^{j-1}, T^{j-1}) \text{ for all } 0 < j \leq \ell, \\ r_0 &= r, & r_j &= \psi_{x_j}^p(r_{j-1}) \text{ for all } 0 < j \leq \ell. \end{aligned}$$

Since r is an efficient k -cut rearrangement of (S, T) and (S, T) is k -cyclic, Proposition 3 implies that 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 3, 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 an efficient $(k + p)$ -cut rearrangement of $\Phi_X^p(S, T)$. The relationship $\Phi_X^p(r(S, T)) = r'(\Phi_X^p(S, T))$ also follows from Lemma 3.

The converse direction is proven similarly using Lemma 4, 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 $\psi_{x_j}^p$, where it drops to k and then remains constant again (so the condition $k' \in \{k, k + p\}$ in Lemma 4 is indeed satisfied). \square

Lemma 6. *If (S, T) is k -cyclic with sample X , then (S, T) is k -efficiently sortable if, and only if, $\Phi_X^p(S, T)$ is k -efficiently sortable.*

Proof. This is a direct application of Lemma 5: a sequence of efficient k -cut rearrangements of (S, T) translates into a sequence of efficient $(k + p)$ -cut rearrangements of $\Phi_X^p(S, T)$ through function Ψ_X^p (note that X remains a sample of (S, T) throughout the sequence of rearrangements). \square

Lemma 7. *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. [12], 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 if, and only if, $\Phi_X^p(S, T)$ is k -efficiently sortable by Lemma 6, and $\Phi_X^p(S, T)$ 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). \square

Theorem 7 is a corollary of Lemma 7, since a k -cyclic pair (S, T) is k -efficiently sortable if, and only if, S can be rearranged into T with at most $\frac{d_b(S,T)}{k}$ k -cut rearrangements (Proposition 1).

3.2. Hardness for Constant Number of Rearrangements

By Theorem 7, SMCR is NP-hard for constant k . We now show that the case with constant ℓ is NP-hard for all $\ell \geq 2$. Since for $\ell = 1$ SMCR is the same as computing the breakpoint distance between permutations, we thus obtain a complexity dichotomy with respect to ℓ .

Theorem 8. SMCR in permutations is NP-hard for any constant $\ell \geq 2$.

We refer to the notions of *breakpoint graph*, *cycles* and *efficient rearrangements* introduced in Section 3.1.1 of the proof of Theorem 7, as well as their first properties (Propositions 1 to 4). We first give an informal outline of our reduction for $\ell = 2$: we show that it is NP-hard to check if an input with $2k$ breakpoints can be solved with two k -cut rearrangements (i.e., can be sorted efficiently). We build two permutations whose cycle graph can be decomposed into cycles of specific sizes, and we show that an efficient sorting scenario needs to satisfy two types of constraints: (i) each cycle must be solved entirely in the first or in the second rearrangement, so the sum of cycle size for each rearrangement must be exactly k , and (ii) some cycles must satisfy precedence constraints, i.e., we generalize the notion of *ties* by building tuples of cycles (C_1, \dots, C_p, C') , where some C_i must be solved before or during the same k -cut rearrangement as C' . We build an instance with a large cycle that needs to be solved during the first rearrangement by constraint (i), and that forces one to solve a non-trivial set of cycles in the same rearrangement by constraint (ii).

Proof. We present a reduction from EXACT 3-SET COVER, where we are given a collection of sets $\mathcal{X} = \{X_1, X_2, \dots, X_n\}$, all of size 3 over a universe U of size $3p$, and ask whether there are p sets in \mathcal{X} whose union is exactly U . Further assume that $7n = 8p$. This is without loss of generality: indeed, let $\delta = 8p - 7n$. If $\delta > 0$, duplicate set X_1 $\lceil \frac{\delta}{7} \rceil$ times, so n increases by this quantity to become n' , and p is unchanged, so δ decreases by $7\lceil \frac{\delta}{7} \rceil$, and now $-7 \leq 8p - 7n' \leq 0$. If $\delta < 0$, add $3|\delta|$ new elements to the universe, and $|\delta|$ sets to \mathcal{X} , covering exactly those new elements, so both n and p increase by $|\delta|$ and become respectively n_1 and p_1 , yielding $8p_1 - 7n_1 = 8p + 8|\delta| - 7n - 7|\delta| = \delta + |\delta| = 0$.

The following reduction is for $\ell = 2$; we show how to extend it to any larger ℓ at the end of the proof. See Figure 6 for an illustration. Given an instance of EXACT 3-SET COVER, we build an alphabet of size $8n + 6p + 2$, with symbols $x_{i,q}$ for $1 \leq i \leq n$ and $1 \leq q \leq 8$, and y_i for $1 \leq i \leq 2p + 2$. Given $u \in U$, let i_1, \dots, i_k be the indices, such that $u \in X_{i_j}$ for all $1 \leq j \leq k$. Furthermore, let p_j be the index of u in X_j (i.e., 1, 2 or 3). Write S_u for the following string:

$$S_u = x_{i_1,2p_1+1} x_{i_1,2p_1} \quad x_{i_2,2p_2+1} x_{i_2,2p_2} \quad \dots \quad x_{i_k,2p_k+1} x_{i_k,2p_k}.$$

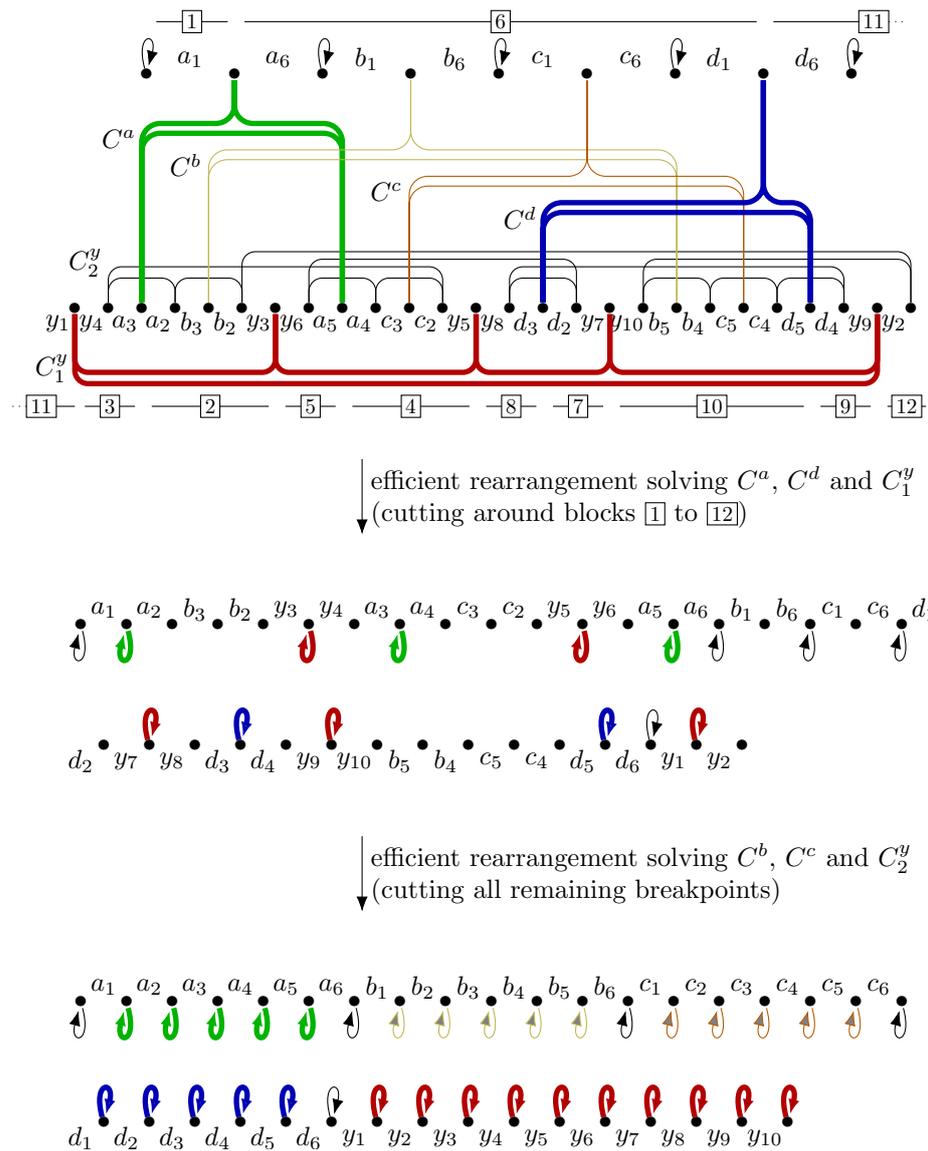


Figure 6. Illustration of the reduction from a simplified version of EXACT COVER, with size-2 sets over a size-4 universe $U = \{1, 2, 3, 4\}$: $A = \{1, 2\}$, $B = \{1, 4\}$, $C = \{2, 4\}$, $D = \{3, 4\}$ (for lighter indices we use a, b, c, d instead of x_1, x_2, x_3, x_4 ; note also that we have a_i for $1 \leq i \leq 6$ instead of $1 \leq i \leq 8$ because of the smaller sets). All six non-trivial cycles of $\mathcal{C}(S, T)$ are depicted on the top permutation: $C^a = \{a_1, a_3, a_5\}$, C^b , etc. In this example, (S, T) admits an efficient 11-cut rearrangement corresponding to an exact set cover of U : cut all breakpoints in bold cycles C^a, C^d, C_1^y , followed by an efficient 19-cut rearrangement solving the remaining breakpoints (in the actual reduction, we consider instances where $7n = 8p$ to enforce that both efficient rearrangements have the same size k).

We now build permutations S and T :

$$S = x_{1,1}x_{1,8} x_{2,1}x_{2,8} \dots x_{n,1}x_{n,8} \quad y_1 y_4 S_1 y_3 y_6 S_2 y_5 \dots y_{6p-1} y_{6p+2} S_{3p} y_{6p+1} y_2$$

$$T = x_{1,1}x_{1,2} \dots x_{1,8} x_{2,1}x_{2,2} \dots x_{2,8} \dots x_{n,1}x_{n,2} \dots x_{n,8} \quad y_1 y_2 \dots y_{6p+2}$$

where the suffix of S is the concatenation of $y_{2i-1}y_{2i+2}S_i$ for every $1 \leq i \leq 3p$, followed by $y_{6p+1}y_2$. Finally, we set $k = 7p + 1$. Note that strings S and T have $n + 1$ adjacencies (0, and each $x_{i,8}$, $1 \leq i \leq n$), so they have $(8n + 6p + 2) + 1 - (n + 1) = 7n + 6p + 2 = 14p + 2 = 2k$ breakpoints. We compute the cycles of $\mathcal{C}(S, T)$. To this end, we describe $n + 2$ cycles

(denoted $C_i^x, 1 \leq i \leq n, C_1^y$ and C_2^y), and check that their total size is indeed equal to the number of breakpoints.

For each $1 \leq i \leq n$, let C_i^x be cycle $x_{i,1} \rightarrow x_{i,3} \rightarrow x_{i,5} \rightarrow x_{i,7} \rightarrow x_{i,1}$. There are n such size-4 cycles, for a total size of $4n$ breakpoints. Let C_1^y be the cycle $y_1 \rightarrow y_{6p+1} \rightarrow y_{6p-1} \rightarrow \dots y_5 \rightarrow y_3 \rightarrow y_1$. It has size $3p + 1$. The remaining arcs in substring $y_{2u+2}S_u y_{2u+1}$ are $y_{2u} \rightarrow x_{i_k,2p_k}, x_{i_j,2p_j} \rightarrow x_{i_{j-1},2p_{j-1}}$ for $2 \leq j \leq k$, and $x_{i_1,2p_1} \rightarrow y_{2u+2}$. Furthermore, $y_{6p+2} \rightarrow y_2$, so all these arcs form a single cycle, C_2^y of length $3n + 3p + 1$ ($3n$ for all $x_{i,2q}, q \in \{1, 2, 3\}$, plus $3p + 1$ for all $y_{2q}, 1 \leq q \leq 3p + 1$). The total size of the cycles above is $(4n) + (3p + 1) + (3n + 3p + 1) = 7n + 6p + 2$, so all breakpoints have been accounted for. Clearly, the reduction can be performed in polynomial time; we now prove that the reduction is correct, i.e.:

S can be transformed into T with $2k$ -cut rearrangements $\Leftrightarrow \mathcal{X}$ has an exact cover of U .

(\Leftarrow) Given an exact cover of U , we show that cutting all breakpoints in cycles C_1^y and C_i^x where x_i is in the set cover yields a k -cut rearrangement solving k breakpoints in S . In other words, we first cut the following breakpoints: for each i such that X_i is in the cover, cut between $x_{i,1}$ and $x_{i,8}$, as well as between $x_{i,2q+1}$ and $x_{i,2q}$ for $1 \leq q \leq 3$. Furthermore, cut after y_{2i+1} for each $0 \leq i \leq 3p$. There are indeed $4p + 3p + 1 = k$ cuts. At this point, each substring $y_{2u+2}S_u y_{2u+1}$ is cut into two strings $y_{2u+2}S'_u x_{j,2p+1}$, and $x_{j,2p}S''_u y_{2u+1}$ (where j is the index of the only set X_j in the cover, such that $u \in X_j$) merge them together in reverse order, i.e., create string $X_{j,p} = x_{j,2p}S''_u y_{2u+1} y_{2u+2} S'_u x_{j,2p+1}$ (note that breakpoint y_{2u+1} is now solved). Furthermore, we merge the prefix ending in y_1 with y_2 , solving an additional breakpoint. Finally, for each j , such that X_j is in the cover, insert strings $X_{j,1}, X_{j,2}, X_{j,3}$ between $x_{j,1}$ and $x_{j,8}$. This step solves breakpoints $x_{j,1}, x_{j,3}, x_{j,5}$ and $x_{j,3}$ for each j in the cover, for a total of $4p$ breakpoints solved. So, overall, k breakpoints of S can be solved with a single k -cut rearrangement. Note that the resulting S' thus has k remaining breakpoints with T : they can all be solved with a single k -cut rearrangement, so S can be sorted into T in only 2 steps.

(\Rightarrow) Permutation S can be sorted into T with $\ell = 2k$ -cut rearrangements if, and only if, it can be sorted using two efficient k -cut rearrangements (Proposition 1). Each of these solves a union of cycles of total size k (Proposition 3). In other words, some cycles (of total size k) are solved first (i.e., with the first rearrangement), the other cycles (of total size k as well) are solved second. Note that all cycles C_i^x together have a size of $4n < k$, so at least one cycle among C_1^y, C_2^y is solved first. They are too large to be solved together, so in fact exactly one of them is solved first, the other second. We extend the notion of being solved first to breakpoints and arcs appearing in the cycles solved first. The following result adapts Proposition 4 to our setting.

Claim 2. *If an arc $u \rightarrow v$ such that v occurs before u in S is solved first, then some breakpoint strictly between v and u in S is solved first as well.*

Proof. Let $vw_1 \dots w_k u$ be the substring of S from v to u . Then, T has uw_1 as a consecutive pair. Solving breakpoint u implies that the rearrangement puts w_1 just after u , which is only possible if there is at least one cut point between w_1 and u in S . \square

Consider any $u \in U$, and element $x_{u,3}$. This element appears in some substring $zx_{u,3}, x_{u,2}$ of S (depending on the position of $x_{u,3}$, z is either of the form y_{2q} or $x_{u',2p}$), and C_2^y has an arc $x_{u,2} \rightarrow z$. By the lemma above, if C_2^y is solved first, then the only breakpoint between z and $x_{u,2}$, (namely, $x_{u,3}$) must be solved first, and the cycle C_u^x containing it as well. So, C_2^y cannot be solved first, since otherwise, all cycles C^x would be solved first as well, for a total size of $7n + 3p + 1 > 7p + 1 = k$.

Thus, C_1^y is solved first, as well as $\frac{k - |C_1^y|}{4} = p$ cycles of the form C_i^x . Write P for the size- p subset of $\{1, \dots, n\}$, such that $i \in P$, if, and only if, C_i^x is solved first. We show that

$\cup_{i \in P} X_i = U$, i.e., P is a solution to the set cover instance. Indeed, let $u \in U$. Recall that S contains substring $y_{2u-1}y_{2u+2}S_u y_{2u+1}$, with arc $y_{2u+1} \rightarrow y_{2u-1}$ in cycle C_1^y . Thus, some other breakpoint z of this substring must be solved first. We have $z \neq y_{2u+2}$ and $z \neq x_{i,2q}$, since otherwise $z \in C_2^y$, which is solved second. Thus $z = x_{i,2q+1}$ for some i such that $u \in X_i$, and since z is solved first, we have $i \in P$, which concludes the hardness proof for $\ell = 2$.

For higher values of ℓ , the following is a reduction of SMCR for ℓ to SMCR for $\ell + 1$: given permutations S, T and an integer k (assuming S and T have $k\ell$ breakpoints), let $S' = S \cdot z_1 \dots z_k$ and $T' = T \cdot z_k \dots z_1$. Then, breakpoints (z_1, \dots, z_k) form a size- k cycle: any efficient sorting of (S, T) must use one step to exclusively sort the breakpoints in this cycle. Thus, (S, T) can be solved with ℓ many k -cut rearrangements if, and only if, (S', T') can be solved with $\ell + 1$ many k -cut rearrangements. \square

In many settings, we encounter a class of *Center* problems, which can be generally defined as follows: given a set of objects S_1, \dots, S_m and a radius k , find an object S^* at distance at most k from each S_i . In particular, in the context of strings, depending on the distance measure, this problem is called **CONSENSUS STRING** (using the Hamming distance) or **CENTER STRING** (using the Edit distance). Problems **LONGEST COMMON SUBSEQUENCE** and **SHORTEST COMMON SUPERSEQUENCE** also fit in this setting, where (non-symmetrical) distances count 1 for deletions (resp. insertions), and forbid all other operations. See [11] for a review of results on these and related problems. Applying this framework to permutations, it is natural to consider the **BREAKPOINT CENTER** problem: given permutations (S_1, \dots, S_m) and an integer k , find a permutation S^* such that the breakpoint distance between each S_i and S^* is at most k . Note that this is slightly different from the **BREAKPOINT MEDIAN** problem [19], usually defined over $m = 3$ permutations, which aims to minimize the sum of distances rather than the maximum. Clearly, for $m = 2$, (S_1, S_2) have a breakpoint center S^* at distance k if, and only if, S_2 can be obtained from S_1 using two k -cut rearrangements (using S^* as an intermediary step). Thus, Theorem 8 can be restated as follows.

Corollary 1. *The BREAKPOINT CENTER problem is NP-hard, even when restricted to two input permutations.*

3.3. Fixed-Parameter Tractability and Approximability

We now show that input instances with bounded k and ℓ can be solved efficiently. More precisely, we show that SMCR in permutations is fixed-parameter tractable when parameterized by $k + \ell$. In fact, SMCR admits a kernel with a polynomial size bound.

Theorem 9. *SMCR in permutations admits a kernel of size $2k\ell + 2$.*

Proof. The kernel is based on the following reduction rule: if there is a duo (a, b) (i.e., a substring ab) in both S and T , then remove b from S and T . Before we show the correctness, observe that the exhaustive application of the rule indeed gives the desired result: any YES-instance that is reduced exhaustively with respect to the above rule has at most $4k\ell + 2$ letters: we must cut inside every duo in S . Overall, we may create at most $k\ell$ cuts via ℓ many k -cut rearrangements. Hence, if S has more than $k\ell$ duos, then (S, T) is a NO-instance. Thus, after applying the rule exhaustively, we may replace any instance with more than $k\ell + 1$ letters in S (and thus in T) by a trivial NO-instance of constant size. Thus, the correctness of the rule remains to be proven. 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 $(S = S_1, S_2, \dots, S_{\ell+1} = T)$, such that S_{i+1} can be obtained from S_i via one k -cut rearrangement. Removing b from S_i gives a sequence $(S'_1, S'_2, \dots, S'_{\ell+1} = T)$ 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 $(S' = S'_1, S'_2, \dots, S'_{\ell+1} = T')$, 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 $(S = S_1, S_2, \dots, S_{\ell+1} = T)$ such that S'_{i+1} can be obtained from S'_i via one k -cut rearrangement. \square

In addition, we show that for arbitrary values of k , SMCR has a polynomial-time approximation algorithm. 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 10. 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 $T' = id_n$, where id_n denotes the identity permutation of $\{1, \dots, n\}$. Let $k' = \lfloor \frac{k}{2} \rfloor$. The algorithm consists of iterating the following three steps, starting from S' :

- (a) rewrite S' , by contracting adjacencies so as to obtain a permutation containing no adjacencies,
- (b) cut around (i.e., right before and right after) the first k' elements $1, 2, 3, \dots, k'$ 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 are presented for 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 $\ell' \leq 2\ell$. \square

4. Conclusions

We introduced SORTING BY MULTI-CUT REARRANGEMENTS, a generalization of usual genome rearrangement problems that, however, does not include reversals. We discussed its classical computational complexity (P vs. NP-hard) and its membership in FPT with respect to the parameters k and ℓ . For this, we distinguished the case where S and T are permutations from the case where S and T are strings.

The obvious remaining open problem is the one indicated in Table 1, namely the FPT status of SMCR with respect to parameter $\ell + k$ in strings (including cases where one of k or ℓ is constant, with $\ell > 1$). Another question is whether the approximation factor of 2 of Theorem 10 can be improved. We also recall that we conjecture SMCR to be NP-hard for any $k \geq 5$ in binary strings. It would also be interesting to better understand the comparative roles of k and ℓ in SMCR, for instance by studying the following question: assuming k is increased by some constant c , what impact does this have on the optimal distance?

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 , as studied, for instance, by Radcliffe et al. [13] in the case of reversals.

Author Contributions: Conceptualization, L.B., G.F., G.J. and C.K.; methodology, L.B., G.F., G.J. and C.K.; validation, L.B., G.F., G.J. and C.K.; formal analysis, L.B., G.F., G.J. and C.K.; writing—original draft preparation, L.B., G.F., G.J. and C.K.; writing—review and editing, L.B., G.F., G.J. and C.K.; supervision, G.F. All authors have read and agreed to the published version of the manuscript.

Funding: This research was partially funded and supported by the PHC Procope grant number 17746PC (G.F. and G.J.), and by the DAAD Procope grant number 57317050 (C.K.).

Data Availability Statement: Not Applicable, the study does not report any data.

Conflicts of Interest: The authors declare no conflict of interest.

Abbreviations

The following abbreviations are used in this manuscript:

SMCR	Sorting by Multi-Cut Rearrangements
MCSP	Minimum Common String Partition
FPT	Fixed-Parameter Tractable

References

1. Bafna, V.; Pevzner, P.A. Genome Rearrangements and Sorting by Reversals. *SIAM J. Comput.* **1996**, *25*, 272–289. [[CrossRef](#)]
2. Bafna, V.; Pevzner, P.A. Sorting by Transpositions. *SIAM J. Discret. Math.* **1998**, *11*, 224–240. [[CrossRef](#)]
3. Christie, D.A. Sorting Permutations by Block-Interchanges. *Inf. Process. Lett.* **1996**, *60*, 165–169. [[CrossRef](#)]
4. Fertin, G.; Labarre, A.; Rusu, I.; Tannier, E.; Vialette, S. *Combinatorics of Genome Rearrangements*; Computational Molecular Biology; MIT Press: Cambridge, MA, USA, 2009.
5. Stephens, P.J.; Greenman, C.D.; Fu, B.; Yang, F.; Bignell, G.R.; Mudie, L.J.; Pleasance, E.D.; Lau, K.W.; Beare, D.; Stebbings, L.A.; et al. Massive Genomic Rearrangement Acquired in a Single Catastrophic Event during Cancer Development. *Cell* **2011**, *144*, 27–40. [[CrossRef](#)] [[PubMed](#)]
6. Pellestor, F.; Gatinois, V. Chromoanagenesis: A piece of the macroevolution scenario. *Mol. Cytogenet.* **2020**, *13*, 3. [[CrossRef](#)] [[PubMed](#)]
7. Hannenhalli, S.; Pevzner, P.A. Transforming Cabbage into Turnip: Polynomial Algorithm for Sorting Signed Permutations by Reversals. *J. ACM* **1999**, *46*, 1–27. [[CrossRef](#)]
8. Bulteau, L.; Fertin, G.; Jean, G.; Komusiewicz, C. Sorting by Multi-cut Rearrangements. In Proceedings of the SOFSEM 2021: Theory and Practice of Computer Science—47th International Conference on Current Trends in Theory and Practice of Computer Science, Bolzano-Bozen, Italy, 25–29 January 2021; pp. 593–607.
9. Downey, R.; Fellows, M.R. *Fundamentals of Parameterized Complexity*; Springer: London, UK, 2013.
10. Cygan, M.; Fomin, F.V.; Kowalik, L.; Lokshtanov, D.; Marx, D.; Pilipczuk, M.; Pilipczuk, M.; Saurabh, S. *Parameterized Algorithms*; Springer: London, UK, 2015.
11. Bulteau, L.; Hüffner, F.; Komusiewicz, C.; Niedermeier, R. Multivariate algorithmics for NP-hard string problems. *Bull. Eur. Assoc. Theor. Comput. Sci.* **2014**, *114*. Available online: <https://hal.archives-ouvertes.fr/hal-01260610/document> (accessed on 27 May 2021).
12. Bulteau, L.; Fertin, G.; Rusu, I. Sorting by Transpositions Is Difficult. *SIAM J. Discret. Math.* **2012**, *26*, 1148–1180. [[CrossRef](#)]
13. Radcliffe, A.J.; Scott, A.D.; Wilmer, E.L. Reversals and transpositions over finite alphabets. *SIAM J. Discret. Math.* **2005**, *19*, 224–244. [[CrossRef](#)]
14. Christie, D.A. Genome Rearrangement Problems. Ph.D. Thesis, University of Glasgow, Glasgow, UK, 1998.
15. Lin, Y.C.; Lu, C.L.; Chang, H.Y.; Tang, C.Y. An efficient algorithm for sorting by block-interchanges and its application to the evolution of vibrio species. *J. Comput. Biol.* **2005**, *12*, 102–112. [[CrossRef](#)] [[PubMed](#)]
16. Kolman, P.; Goldstein, A.; Zheng, J. Minimum Common String Partition Problem: Hardness and Approximations. *Electron. J. Comb.* **2005**, *12*. [[CrossRef](#)]
17. Garey, M.R.; Johnson, D.S. *Computers and Intractability: A Guide to the Theory of NP-Completeness*; W. H. Freeman & Co.: New York, NY, USA, 1979.
18. Bulteau, L.; Komusiewicz, C. Minimum Common String Partition Parameterized by Partition Size Is Fixed-Parameter Tractable. In Proceedings of the Twenty-Fifth Annual ACM-SIAM Symposium on Discrete Algorithms, SODA 2014, Portland, OR, USA, 5–7 January 2014; pp. 102–121.
19. Sankoff, D.; Blanchette, M. The Median Problem for Breakpoints in Comparative Genomics. In Proceedings of the Computing and Combinatorics, Third Annual International Conference, COCOON '97, Shanghai, China, 20–22 August 1997; Springer: London, UK, 1997; Volume 1276, pp. 251–264.