

Approximability of Longest Run Subsequence and Complementary Minimization Problems

Yuichi Asahiro 


Kyushu Sangyo University, Fukuoka, Japan

Jesper Jansson 


Kyoto University, Kyoto, Japan

Sichen Lu 


University of Alberta, Edmonton, Canada

Hiroataka Ono 


Nagoya University, Nagoya, Japan

Shunichi Tanaka 


Kyushu Institute of Technology, Iizuka, Japan

Mingyang Gong 

University of Alberta, Edmonton, Canada

Guohui Lin 

University of Alberta, Edmonton, Canada

Eiji Miyano 

Kyushu Institute of Technology, Iizuka, Japan

Toshiki Saitoh 

Kyushu Institute of Technology, Iizuka, Japan

Abstract

We study the polynomial-time approximability of the LONGEST RUN SUBSEQUENCE PROBLEM (LRS for short) and its complementary minimization variant MINIMUM RUN SUBSEQUENCE DELETION PROBLEM (MRSD for short). For a string $S = s_1 \cdots s_n$ over an alphabet Σ , a *subsequence* S' of S is $S' = s_{i_1} \cdots s_{i_p}$, such that $1 \leq i_1 < i_2 < \dots < i_p \leq |S|$. A *run of a symbol* $\sigma \in \Sigma$ in S is a maximal substring of consecutive occurrences of σ . A *run subsequence* S' of S is a subsequence of S in which every symbol $\sigma \in \Sigma$ occurs in at most one run. The *co-subsequence* $\overline{S'}$ of the subsequence $S' = s_{i_1} \cdots s_{i_p}$ in S is the subsequence obtained by deleting all the characters in S' from S , i.e., $\overline{S'} = s_{j_1} \cdots s_{j_{n-p}}$ such that $j_1 < j_2 < \dots < j_{n-p}$ and $\{j_1, \dots, j_{n-p}\} = \{1, \dots, n\} \setminus \{i_1, \dots, i_p\}$. Given a string S , the goal of LRS (resp., MRSD) is to find a run subsequence S^* of S such that the length $|S^*|$ is maximized (resp., the number $|\overline{S^*}|$ of deleted symbols from S is minimized) over all the run subsequences of S . Let k be the maximum number of symbol occurrences in the input S . It is known that LRS and MRSD are APX-hard even if $k = 2$. In this paper, we show that LRS can be approximated in polynomial time within factors of $(k+2)/3$ for $k = 2$ or 3 , and $2(k+1)/5$ for every $k \geq 4$. Furthermore, we show that MRSD can be approximated in linear time within a factor of $(k+4)/4$ if k is even and $(k+3)/4$ if k is odd.

2012 ACM Subject Classification Theory of computation \rightarrow Design and analysis of algorithms

Keywords and phrases Longest run subsequence, minimum run subsequence deletion, approximation algorithm

Digital Object Identifier 10.4230/LIPIcs.WABI.2025.3

Funding The work was partially supported by the NSERC Canada, JSPS KAKENHI Grant Numbers JP20H05967, JP22H00513, JP22K11915, JP24H00697, JP24K02898, JP24K02902, JP24K14827, and JP25K03077, and CRONOS Grant Number JPMJCS24K2.

1 Introduction

Scaffolding is one of the key informatics processes in DNA sequencing. DNA sequencing is generally carried out through the following steps: (i) Tens to hundreds of millions of DNA fragments extracted from random positions are read via shotgun sequencing, (ii) the extracted random fragments (reads) are assembled into a series of contiguous sequences (contigs) using an assembly algorithm, and (iii) finally, the contigs are arranged in the correct order based on certain criteria. This step (iii) is called scaffolding, which serves as the original motivation of this study. One common approach to scaffolding is to rearrange contigs by comparing multiple incomplete assemblies of related samples (see [10] for example).



© Yuichi Asahiro, Mingyang Gong, Jesper Jansson, Guohui Lin, Sichen Lu, Eiji Miyano, Hiroataka Ono, Toshiki Saitoh, and Shunichi Tanaka;

licensed under Creative Commons License CC-BY 4.0

25th International Conference on Algorithms for Bioinformatics (WABI 2025).

Editors: Broňa Brejová and Rob Patro; Article No. 3; pp. 3:1–3:20



Leibniz International Proceedings in Informatics

Schloss Dagstuhl – Leibniz-Zentrum für Informatik, Dagstuhl Publishing, Germany

The formulation of contig rearrangement from multiple incomplete assemblies in the scaffolding phase as a string processing problem by Schrinner et al. [11, 12] is known as the LONGEST RUN SUBSEQUENCE PROBLEM (LRS). Let Σ be a finite alphabet of symbols, and $|\Sigma| = m$. A *string* $S = s_1 \cdots s_n$ is a sequence of n *characters*, each of which is a symbol in Σ . Two or more characters in S can be the same symbol in Σ . For a string $S = s_1 \cdots s_n$, $|S|$ denotes the length of S , i.e., $|S| = n$. For two strings S_1 and S_2 , $S_1 \circ S_2$ denotes the concatenation of S_1 and S_2 . A *subsequence* S' of S is a sequence $S' = s_{i_1} \cdots s_{i_p}$, such that $1 \leq i_1 < i_2 < \dots < i_p \leq |S|$. Let $S[i]$ denote the character of S in the i th position for $1 \leq i \leq |S|$, and $S[i, j]$ denote the substring of S that starts from the i th position and ends at the j th position. A σ -*run* in S is a substring $S[i, j]$ such that $S[\ell] = \sigma$, for any $\ell = i, i+1, \dots, j$, but $S[i-1] \neq \sigma$ and $S[j+1] \neq \sigma$. Given a string S on alphabet Σ , a *run subsequence* S' of S is a subsequence of S in which every symbol $\sigma \in \Sigma$ occurs in at most one run. For a string $S = abbbaab$, for example, the substring $S[2, 4]$ is a b -run, and $aaab$ is a run subsequence of S .

► **Problem 1** (LONGEST RUN SUBSEQUENCE PROBLEM, LRS). *Given a string S in Σ^n , the goal of LRS is to find a longest run subsequence S^* of S , i.e., every $\sigma \in \Sigma$ occurs in at most one run in S^* and the length $|S^*|$ is maximized over all the run subsequences of S .*

For the string $S = abbbaab$, the longest run subsequence S^* of S is $bbbaa$ of length five. If the maximum number of occurrences of each symbol in the input string S is bounded by k , then the problem is called the k -LONGEST RUN SUBSEQUENCE PROBLEM (k -LRS). One sees that 1-LRS is trivial since the length of all the runs in the input string S is one, and thus the input S itself is the optimal run subsequence. Unfortunately, Schrinner et al. [12] showed that LRS is generally NP-hard. Subsequently, Dondi and Sikora [5] showed 2-LRS is APX-hard, while, as a positive result, they provided a polynomial-time k -approximation algorithm for k -LRS. Recently, Asahiro et al. [2] improved the approximation ratio to $(k+1)/2$ for k -LRS, and have shown that for the case $k = 2$, a better approximation ratio $4/3$ than $(k+1)/2 = 3/2$ is achieved.

In this paper, we first derive further improved approximability results for k -LRS:

► **Theorem 1.** *k -LRS can be approximated in $O(mn^2)$ time within factors of $(k+2)/3$ for $k = 2$ or 3, and $2(k+1)/5$ for every $k \geq 4$.*

This paper also considers the complementary minimization variant of LRS, called the MINIMUM RUN SUBSEQUENCE DELETION problem (MRSD). The *co-subsequence* $\overline{S'}$ of the subsequence $S' = s_{i_1} \cdots s_{i_p}$ in S is the subsequence obtained by deleting all the characters in S' from S , i.e., $\overline{S'} = s_{j_1} \cdots s_{j_{n-p}}$ such that $j_1 < j_2 < \dots < j_{n-p}$ and $\{j_1, \dots, j_{n-p}\} = \{1, \dots, n\} \setminus \{i_1, \dots, i_p\}$. For example, consider $S = abcde$. Then, for a subsequence $S' = abd$, the co-subsequence $\overline{S'}$ of S' is $\overline{S'} = ce$. Note that for a subsequence S' of a string S , $\overline{S'}$ is not unique unless we specify position of each character of S' in S : for $S = ababa$ and $S' = aa$ (without indices from which these a 's come), candidates of $\overline{S'}$ is bba , bab , and abb of the same length three. As will be seen in the following, only the number of deleted characters is important in some cases, but we often need to take care of from where a character is deleted.

► **Problem 2** (MINIMUM RUN SUBSEQUENCE DELETION PROBLEM, MRSD). *Given a string S in Σ^n , the goal of MRSD is to find a run subsequence S^* of S such that the number $|\overline{S^*}|$ of deleted symbols from S is minimized over all the run subsequence of S .*

Similarly to k -LRS, if the maximum number of occurrences of each symbol in the input string S is bounded by k , the problem is called the k -MINIMUM RUN SUBSEQUENCE DELETION PROBLEM (k -MRSD). Since the run subsequence obtained by minimizing the

number of deletions in MRSD corresponds exactly to the longest run subsequence in LRS, LRS and MRSD are essentially equivalent as decision problems. However, due to the difference in the objective functions, MRSD may exhibit different characteristics from LRS in terms of approximability. Thus, a natural question arises: Which problem is easier/harder to approximate, or are they equally hard?

To gain insight into this question, we consider two examples of problem pairs that are essentially equivalent as decision problems but differ in their objective functions, similar to LRS and MRSD. The first example is the pair of MAX-2SAT and its deletion variant, MIN-2SAT DELETION. It is known that 2SAT can be solved in polynomial time [3]. However, in the case where the instance is unsatisfiable, the objective of MAX-2SAT is to find a truth assignment that maximizes the number of satisfied clauses (a maximization problem). In contrast, the objective of MIN-2SAT DELETION is to minimize the number of unsatisfied clauses (a minimization problem). For these problems, the following results are known: The maximization version, MAX-2SAT, admits a 1.0638-approximation algorithm [9], but it is NP-hard to approximate within a factor of 1.0476 [6]. On the other hand, the best known approximation ratio for the minimization version, MIN-2SAT DELETION, is $O(\sqrt{\log n})$ [1], and it is known to be NP-hard to approximate within a factor of 1.36067[4]. Thus, intuitively, the maximization version is easier to approximate than the deletion version for these two problems.

Another example is the pair consisting of the MAXIMUM INDEPENDENT SET PROBLEM (MaxIS) and the MINIMUM VERTEX COVER PROBLEM (MinVC). For any graph $G = (V, E)$ and any independent set $IS \subseteq V$ of G , $V \setminus IS$ forms a vertex cover. Thus, MinVC can be seen as the complementary minimization variant of MaxIS. For these problems, the following results are known: MaxIS is NP-hard to approximate within a factor of $n^{1-\varepsilon}$ for any $\varepsilon > 0$ [13]. In contrast, MinVC is known to admit a $2 - \Theta(1/\sqrt{\log n})$ -approximation algorithm [7]. Thus, contrary to the previous pair, in this case, the maximization version is harder to approximate than the deletion version.

As the second contribution, the paper investigates the approximability of k -MRSD:

► **Theorem 2.** *k -MRSD can be approximated in linear time within a factor of $(k+4)/4$ for even k , and $(k+3)/4$ for odd k .*

Namely, unlike the two aforementioned examples, LRS and MRSD can be considered as problems that currently share similar approximation ratios of $O(k)$. We remark that the basic strategies of the proposed approximation algorithms for MRSD are very similar to those for LRS, but the analyses of the approximation ratios are quite different.

Notation. For each symbol $\sigma \in \Sigma$, σ^h and $\ell_{\max}(\sigma)$ denote a length- h σ -run and the length of the longest σ -run in the input string S , respectively. Let $\text{occ}(\sigma)$ be the number of occurrences of σ in the input string S . Let $\text{occ}_{\max}(S) = \max_{\sigma \in \Sigma} \text{occ}(\sigma)$. Without loss of generality, we assume that the number of occurrences of each symbol $\sigma \in \Sigma$ in S is at least one, and if it is one, then we say σ is unique.

► **Example 3.** Consider a string $S = \text{babcbcadeggg}$. Then S includes two a^1 , three b^1 , two c^1 , two d^1 , and one e^1 , where each length is one; and a length-3 g -run g^3 . Therefore, the length of the longest run for a , b , c , d , or e is 1, and for g is 3, respectively. That is, $\ell_{\max}(a) = \ell_{\max}(b) = \ell_{\max}(c) = \ell_{\max}(d) = \ell_{\max}(e) = 1$ and $\ell_{\max}(g) = 3$. The number $\text{occ}(b)$ of occurrences of b is three, e is unique, i.e., $\text{occ}(e) = 1$, and $\text{occ}_{\max}(S) = 3$.

2 Approximation algorithms for Longest Run Subsequence

In this section we consider LRS and design approximation algorithms for k -LRS. That is, we assume that the maximum number $\text{occ}_{\max}(S)$ of symbol occurrences in the input S is always bounded by k .

2.1 Preprocessing

We first introduce an inserting operation to preprocess the input string S . For every symbol σ with $\ell_{\max}(\sigma) \geq 3$, we create an auxiliary symbol σ' . The auxiliary alphabet that contains all the auxiliary symbols is denoted as Σ' . The inserting operation inserts a copy of a symbol σ' after the first two consecutive symbols $\sigma\sigma$ in a σ^h with $h \geq 3$. We repeatedly apply the inserting operation until there is no σ^h for any symbol σ and any $h \geq 3$.

► **Operation** (An inserting operation). Given a σ^h with $h \geq 3$ in the string S , the operation inserts a copy of the symbol $\sigma' \in \Sigma'$ after the first two consecutive symbols $\sigma\sigma$.

► **Example 4.** Recall the string S in Example 3 and $\ell_{\max}(g) = 3$. After the preprocessing, g^3 becomes $ggg'g$, and the resulting string is $S' = \text{babcbcadeggg}'g$.

One clearly sees that in the preprocessed string, denoted as S' , $\ell_{\max}(\sigma) \leq 2$ for any symbol $\sigma \in \Sigma$ (and $\ell_{\max}(\sigma') = 1$ for any auxiliary symbol $\sigma' \in \Sigma'$). Let $\Pi \subseteq \Sigma$ denote the subset of symbols σ 's such that $\ell_{\max}(\sigma) = 2$, and then let $\Lambda = (\Sigma \cup \Sigma') \setminus \Pi$.

2.2 The algorithm

We present an algorithm **ALG1** to compute a run-subsequence $\text{ALG1}(S')$ for the preprocessed string S' obtained by applying the above preprocessing to S .

► **Definition 5.** For a symbol $s \in \Lambda$ that is not unique, if every two consecutive occurrences of s in S' are separated by at least two symbols, then s is a *good* symbol. Otherwise, s is a *bad* symbol and every substring of S' in the form of sts , where t is another symbol, is a *bad segment* associated with s .

Using Definition 5, we partition Λ into three subsets Λ_1, Λ_2 and Λ_3 , where Λ_1 contains all the unique symbols, Λ_2 contains all the good symbols, and Λ_3 contains all the bad symbols. One sees that such a partition can be done in $O(|S'|)$ time.

Our algorithm constructs an initial solution for S' and then applies two local search operations to update the solution. During the algorithm **ALG1**, the current solution is always denoted as $\text{ALG1}(S')$. Initially, for each symbol $s \in \Pi \cup \Lambda_1 \cup \Lambda_2$, the algorithm picks its leftmost longest run in S' into $\text{ALG1}(S')$; for each bad symbol $s \in \Lambda_3$, the algorithm picks the leftmost s in the first bad segment associated with s into $\text{ALG1}(S')$. Note that all these picked runs are in the same order as they show up in S' , i.e., $\text{ALG1}(S')$ is a subsequence of S' . We continue to illustrate using the string in Example 4.

► **Example 6.** Consider the string $S' = \text{babcbcadeggg}'g$. We have $\Pi = \{g\}$, $\Lambda_1 = \{e, g'\}$, $\Lambda_2 = \{a\}$, and $\Lambda_3 = \{b, c, d\}$. The initial solution $\text{ALG1}(S')$ is $\text{bacdeg}'g$, which is obtained by picking the symbols in the boxes as follows:

$\boxed{b} \boxed{a} b \boxed{c} b c a \boxed{d} \boxed{e} d \boxed{gg} \boxed{g'} g.$

Observe that if the symbol t after the picked bad symbol s in the associated bad segment sts is not picked, then we can add the second bad symbol s in the bad segment to the solution to increase its length by one. In the sequel, we aim to do this by possibly swapping some picked good symbols with their respective copies at the other places.

To this purpose, we further partition Λ_3 into two subsets Λ'_3 and Λ''_3 , where Λ''_3 contains those bad symbols, each of which has a length-2 run in the solution $ALG1(S')$. At the beginning, $\Lambda'_3 = \Lambda_3$ and $\Lambda''_3 = \emptyset$.

We design two local operations to repeatedly improve the initial solution $ALG1(S')$. The first operation is almost the above observation, while the second goes slightly further to swap a picked symbol.

► **Operation (Local operation-1 for $s \in \Lambda'_3$).** Given a symbol $s \in \Lambda'_3$ in a bad segment sts such that its symbol t is not picked into $ALG1(S')$, the operation replaces s in $ALG1(S')$ by the two copies of s in this bad segment, and moves s from Λ'_3 to Λ''_3 .

► **Operation (Local operation-2 for $s \in \Lambda'_3$).** Given a symbol $s \in \Lambda'_3$ in a bad segment sts such that its symbol $t \in \Lambda_2 \cup \Lambda'_3$ is picked into $ALG1(S')$, the operation finds another occurrence of t in S' that does not break any length-2 runs in $ALG1(S')$, replaces the picked t in $ALG1(S')$ by this occurrence and replaces s in $ALG1(S')$ by the two copies of s in this bad segment, and moves s from Λ'_3 to Λ''_3 .

We remark that while applying the above two local operations, the σ^2 -run for each $\sigma \in \Pi$ and the σ -run for each $\sigma \in \Lambda_1$ are untouched; for each $\sigma \in \Lambda_2 \cup \Lambda'_3$, it appears exactly once in $ALG1(S')$, while for each $\sigma \in \Lambda''_3$, it appears twice in $ALG1(S')$ and they are picked from a bad segment associated with σ . The goal of the local search is to reduce the number of bad symbols in Λ'_3 as much as possible, and the process terminates when none of the two local operations is applicable. A high-level description of the algorithm is as follows:

■ **Algorithm 1** A high-level description of the algorithm **ALG1**.

Input: The sequence S' obtained from preprocessing S .

Output: A subsequence $ALG1(S')$ of S' .

-
- 1: Partition the symbols in S' into four subsets Π , Λ_1 , Λ_2 , and Λ_3 .
 - 2: Construct an initial solution $ALG1(S')$ where for each $\sigma \in \Pi \cup \Lambda_1 \cup \Lambda_2$, pick its leftmost longest σ -run and if $\sigma \in \Lambda_3$ then pick the first σ -run in its first bad segment.
 - 3: Repeatedly apply the local operation-1 and 2 to update $ALG1(S')$ until impossible.
 - 4: Return $ALG1(S')$ as the solution.
-

We examine the time complexity of **ALG1**. First notice that $|S'| \leq 3n/2$ and $|\Sigma'| \leq m$. Therefore, the partition of $\Sigma \cup \Sigma'$ into $\Pi, \Lambda_1, \Lambda_2, \Lambda_3$ can be done in $O(n)$ time. One sees that **ALG1** executes at most $2m$ local operations, and finding a symbol in Λ'_3 to which the local operation-1 is applicable takes $O(n)$ time and finding a symbol in Λ'_3 to which the local operation-2 is applicable takes $O(n^2)$ time. It follows that **ALG1** runs in $O(mn^2)$.

We continue to illustrate using the string in Example 4.

► **Example 7.** Consider the string $S' = babcbcadeggg'g$. From Example 6, the initial solution $ALG1(S')$ is $bacdeggg'$ and $\Lambda_3 = \{b, c, d\}$, so that $\Lambda'_3 = \{b, c, d\}$ and $\Lambda''_3 = \emptyset$.

The local operation-1 is applicable for $c \in \Lambda'_3$ which is associated with only one bad segment cbc , where its b is not picked. Thus $ALG1(S')$ is updated to $baccdeggg'$ by picking the symbols in the boxes as follows:

$$\boxed{b} \boxed{a} b \boxed{c} b \boxed{c} a \boxed{d} \boxed{e} d \boxed{gg} \boxed{g'} g,$$

and $\Lambda'_3 = \{b, d\}$ and $\Lambda''_3 = \{c\}$.

One sees that the local operation-1 is no longer applicable for symbols b and d in Λ'_3 , but then the local operation-2 is applicable to b in the first bad segment bab associated with b , since the second occurrence of a does not break any length-2 runs in $ALG1(S')$. As a result, $ALG1(S')$ is updated to $bbccaedggg'$ by picking the symbols in the boxes as follows:

$$\boxed{b} a \boxed{b} \boxed{c} b \boxed{c} a \boxed{d} \boxed{e} d \boxed{gg} \boxed{g'} g$$

and $\Lambda'_3 = \{d\}$ and $\Lambda''_3 = \{b, c\}$. For the last symbol $d \in \Lambda'_3$, we cannot apply neither the local operation-1 nor the local operation-2, and thus the algorithm terminates.

2.3 Post-processing

We process the achieved solution $ALG1(S')$ to produce a solution $ALG1(S)$ for the input S . Recall that S' is the result of preprocessing S by the inserting operations, each inserts a copy of the auxiliary symbol σ' after the first two consecutive symbols $\sigma\sigma$ in a σ^h with $h \geq 3$. For each auxiliary symbol $\sigma' \in \Sigma'$, we delete its single copy from $ALG1(S')$ and then replace the σ^2 in $ALG1(S')$ by a longest σ -run in the input sequence S . We remark that σ' is either unique or good, and thus it appears exactly once in $ALG1(S')$, and that no symbol of the longest σ -run in the input sequence S breaks any length-2 run in $ALG1(S')$.

We continue to illustrate using the string in Example 3.

► **Example 8.** Consider the string $S = babcbcadeggg$, for which $S' = babcbcadeggg'g$, where g' is inserted by the inserting operation. From Example 7, the achieved solution $ALG1(S')$ for S' is $bbccadegg'$ by picking the symbols in the boxes as follows:

$$\boxed{b} a \boxed{b} \boxed{c} b \boxed{c} a \boxed{d} \boxed{e} d \boxed{gg} \boxed{g'} g,$$

The post-processing gives the solution $ALG1(S) = bbccadegg$ for S by picking the symbols in the boxes as follows:

$$\boxed{b} a \boxed{b} \boxed{c} b \boxed{c} a \boxed{d} \boxed{e} d \boxed{ggg}.$$

2.4 Performance analysis

In this section, we analyze the worst-case ratio of our algorithm. Let $ALG1(S)$ and $OPT(S)$ denote the output sequence by our algorithm and an optimal solution when the input sequence is S , respectively. Let $\alpha(S) = \frac{|OPT(S)|}{|ALG1(S)|}$.

► **Lemma 9.** For an input string S , let S' be the result of preprocessing S by the inserting operation. Then $\alpha(S') \geq \alpha(S)$ is satisfied.

Proof. S is a subsequence of S' . Therefore, $|OPT(S)| \leq |OPT(S')|$.

We next prove $|ALG1(S')| \leq |ALG1(S)|$. Note that for $\sigma' \in \Sigma'$, it corresponds to a symbol σ with $\ell_{max}(\sigma) \geq 3$ in S . If $\ell_{max}(\sigma) = 3$ in S , then σ' is unique in S' . Otherwise, $\ell_{max}(\sigma) \geq 4$ in S , then σ' is a good symbol since in between every two σ' symbols, there is a σ^2 with $\sigma \neq \sigma'$. That is, $\sigma' \in \Lambda_1 \cup \Lambda_2$ and thus the length of σ' in $ALG1(S')$ is exactly one. Therefore, the total length of symbols σ and σ' in $ALG1(S')$ is exactly three. It indicates that $|ALG1(S')| \leq |ALG1(S)|$ since we choose the longest σ -run whose length $\ell_{max}(\sigma)$ is at least three in $ALG1(S)$. Then the lemma is proved. ◀

We remark that since the approximation ratio of the string S' is always worse than the one of S by Lemma 9, we show a bound on the approximation ratio of S' instead of the original input S .

Now, we are ready to prove the worst-case ratio of our algorithm. Let n_i be the number of unique symbols whose length is i in the optimal solution $OPT(S')$ for $i = 0, 1$. Recall that for each symbol $\sigma \in \Lambda_2$, the length of the σ -run in $ALG1(S')$ must be one. We use x_i to denote the number of symbols in Λ_2 whose length is i in $OPT(S')$ for $i = 0, 1, \dots, k$. Similarly, let y_i (z_i , respectively) be the number of symbols in Λ'_3 (Λ''_3 , respectively) whose length is i in $OPT(S')$ for $i = 0, 1, \dots, k$. Lastly, for each symbol $\sigma \in \Pi$, the length of the σ -run in $ALG1(S')$ is two. We denote by t_i the number of symbols in Π whose length is i in $OPT(S')$ for $i = 0, 1, \dots, k$. In conclusion, we have the following two equalities:

$$|OPT(S')| = n_1 + \sum_{i=1}^k i(x_i + y_i + z_i + t_i), \text{ and} \quad (1)$$

$$|ALG1(S')| = n_0 + n_1 + \sum_{i=0}^k (x_i + y_i + 2z_i + 2t_i). \quad (2)$$

Let $\#D$ be the total number of symbols deleted from S' by the optimal algorithm to obtain $OPT(S')$. Notice that if the length of the σ -run is i in $OPT(S')$ and σ is not unique, then the total number of deleted σ symbols is at most $k - i$. Therefore the upper bound on $\#D$ is

$$\#D \leq n_0 + \sum_{i=0}^{k-1} (k - i)(x_i + y_i + z_i + t_i). \quad (3)$$

Then we estimate a lower bound on $\#D$ as follows.

► **Lemma 10.** *For the number of deleted symbols $\#D$, we have*

$$\#D \geq \sum_{i=2}^k \left(2(i-1)x_i + (i-1)(y_i + z_i) + \left(\frac{i}{2} - 1 \right) t_i \right).$$

Proof. Since each symbol $\sigma \in \Lambda_2$ is a good symbol, there exist at least two other symbols in between every two σ -runs. In order to generate a length- i σ -run in $OPT(S')$, at least $2(i-1)$ symbols are deleted.

For each symbol $\sigma \in \Lambda_3$, $\ell_{\max}(\sigma) = 1$ in S' . So, forming a length- i σ -run must delete at least $i-1$ symbols.

Lastly, consider a symbol $\sigma \in \Pi$. Recall that $\ell_{\max}(\sigma) = 2$. It indicates that we must delete at least one symbol to generate every length-3 σ -run. So for each symbol in Π , we must delete at least $\frac{i}{2} - 1$ symbols.

This completes the proof of this lemma. ◀

By Lemma 10 and Eq.(3), we have

$$\sum_{i=2}^k \left(2(i-1)x_i + (i-1)(y_i + z_i) + \left(\frac{i}{2} - 1 \right) t_i \right) \leq n_0 + \sum_{i=0}^{k-1} (k - i)(x_i + y_i + z_i + t_i),$$

which is equivalent to

$$\begin{aligned}
& 2(k-1)x_k + (k-1)(y_k + z_k) + \left(\frac{k}{2} - 1\right)t_k \\
& \leq n_0 + k(x_0 + y_0 + z_0 + t_0) + (k-1)(x_1 + y_1 + z_1 + t_1) \\
& \quad + \sum_{i=2}^{k-1} (k-3i+2)x_i + \sum_{i=2}^{k-1} (k-2i+1)(y_i + z_i) + \sum_{i=2}^{k-1} \left(k - \frac{3i}{2} + 1\right)t_i.
\end{aligned} \tag{4}$$

► **Lemma 11.** *The summation of y_i, z_i satisfies the following inequality:*

$$\sum_{i=0}^k y_i \leq n_0 + n_1 + \sum_{i=0}^k z_i.$$

Proof. The proof appears in Appendix A. ◀

We can show the following theorem:

► **Theorem 1.** *k -LRS can be approximated in $O(mn^2)$ time within factors of $(k+2)/3$ for $k=2$ or 3 , and $2(k+1)/5$ for every $k \geq 4$.*

Proof sketch. Let ℓ be a constant; to get the best worst-case ratio, we will set $\ell = 1/3$ for $k=2, 3$, and $\ell = (2k-3)/(5k-5)$ for $k \geq 4$ later.

By Eq.(4) and Eq.(1), we have

$$\begin{aligned}
& k(x_k + y_k + z_k + t_k) \\
& = 2(k-1)\ell x_k + (k-1)\ell(y_k + z_k) + \left(\frac{k}{2} - 1\right)\ell t_k \\
& \quad + (k - (2k-2)\ell)x_k + (k - (k-1)\ell)(y_k + z_k) + \left(k - \left(\frac{k}{2} - 1\right)\ell\right)t_k \\
& \leq n_0\ell + k\ell(x_0 + y_0 + z_0 + t_0) + (k-1)\ell(x_1 + y_1 + z_1 + t_1) \\
& \quad + \sum_{i=2}^{k-1} (k-3i+2)\ell x_i + \sum_{i=2}^{k-1} (k-2i+1)\ell(y_i + z_i) + \sum_{i=2}^{k-1} \left(k - \frac{3i}{2} + 1\right)\ell t_i \\
& \quad + (k - (2k-2)\ell)x_k + (k - (k-1)\ell)(y_k + z_k) + \left(k - \left(\frac{k}{2} - 1\right)\ell\right)t_k, \text{ and}
\end{aligned} \tag{5}$$

$$\begin{aligned}
|OPT(S')| &= n_1 + \sum_{i=1}^{k-1} i(x_i + y_i + z_i + t_i) + k(x_k + y_k + z_k + t_k) \\
&\leq n_0\ell + n_1 + k\ell(x_0 + y_0 + z_0 + t_0) + ((k-1)\ell + 1)(x_1 + y_1 + z_1 + t_1) \\
&\quad + \sum_{i=2}^{k-1} ((k-3i+2)\ell + i)x_i + \sum_{i=2}^{k-1} ((k-2i+1)\ell + i)(y_i + z_i) \\
&\quad + \sum_{i=2}^{k-1} \left(\left(k - \frac{3i}{2} + 1\right)\ell + i\right)t_i + (k - (2k-2)\ell)x_k \\
&\quad + (k - (k-1)\ell)(y_k + z_k) + \left(k - \left(\frac{k}{2} - 1\right)\ell\right)t_k.
\end{aligned} \tag{6}$$

We discuss two cases of k . When $k = 2, 3$, $\ell = \frac{1}{3}$ and $\frac{k+2}{3} \geq \frac{5k+2}{12}$ and $\frac{k+2}{3} \leq \min\{\frac{2k+1}{3}, \frac{5k+2}{6}\}$. By Lemma 11, Eqs. (6) and (2), the following inequality holds.

$$|OPT(S')| \leq \frac{k+2}{3} |ALG1(S')|. \quad (7)$$

On the other hand, when $k \geq 4$, $\ell = \frac{2k-3}{5k-5}$ and thus $\ell \in [\frac{1}{3}, \frac{1}{2})$. By Lemma 11, we have

$$\sum_{i=0}^k (y_i + z_i) \leq \frac{1}{3}(n_0 + n_1) + \frac{2}{3} \sum_{i=0}^k (y_i + 2z_i). \quad (8)$$

Here we obtain several inequalities that are used in the following. If $2 \leq i \leq k-1$, then $(k-3i+2)\ell + i \leq (k-4)\ell + 2 \leq (k-1)\ell + 1$, and $k - (2k-2)\ell \leq (k-1)\ell + 1$. Moreover, $(k-2i+1)\ell + i \leq (3-k)\ell + k-1 \leq k - (k-1)\ell$, and $(k - \frac{3i}{2} + 1)\ell + i \leq \frac{(5-k)\ell}{2} + k-1 \leq k - (\frac{k}{2} - 1)\ell$. Note that the following inequality holds:

$$(k-1)\ell + 1 \leq k - (k-1)\ell \leq k - \left(\frac{k}{2} - 1\right)\ell \leq 2(k-1)\ell + 2.$$

By Eqs.(8) and (2), Eq.(6) can be simplified into the following inequality:

$$|OPT(S')| \leq \frac{2(k+1)}{5} |ALG1(S')|. \quad (9)$$

Combining Eqs.(7) and (9), the theorem is proved. \blacktriangleleft

We note that the analysis on the approximation ratio of $(k+2)/3$ for $k = 2$ or 3 is strictly tight. However, for the $(2k+2)/5$ ratio of $k \geq 4$, we know only a bad example for which the approximation ratio is $(2k+1)/5$; there remains a slight gap. See Appendix C for details.

3 Simple approximation algorithm for MRSD

In this section, we consider the deletion variant k -MRSD. Again, assume that $occ_{max}(S)$ is bounded by k . As a warm-up, we design a simple approximation algorithm **ALG2** with approximation ratios $(k+2)/2$ if k is even, and $(k+1)/2$ if k is odd:

Algorithm 2 A high-level description of our algorithm **ALG2**.

Input: An input string S in which every symbol $\sigma \in \Sigma$ appears at most k times.

Output: A run subsequence $ALG2(S)$ of S .

-
- 1: For each $\sigma \in \Sigma$, select the leftmost longest σ -run in S .
 - 2: Output the concatenation of all the selected longest runs.
-

That is, **ALG2** deletes all unselected runs for every symbol from S .

► **Example 12.** Consider the input string $S = a^1 b^1 a^2 b^1 c^1 a^2 b^1 c^1 b^1 c^2 b^2$ of length 15 for MRSD. The leftmost longest a -run, b -run, and c -run are $S[3, 4]$, $S[14, 15]$, and $S[12, 13]$, respectively. Therefore, the output subsequence of **ALG2** is $ALG2(S) = a^2 c^2 b^2$, and thus the co-subsequence of $ALG2(S)$ is $\overline{ALG2(S)} = a^1 b^2 c^1 a^2 b^1 c^1 b^1$. Therefore, the number $|\overline{ALG2(S)}|$ of deleted characters from S by our algorithm **ALG2** is nine.

Clearly, **ALG2** can be implemented in linear time. We bound its approximation ratio in the following. Let S be an input string of k -MRSD. Suppose that $OPT(S)$ and $ALG2(S)$ are solutions obtained by an optimal algorithm **OPT** and our algorithm **ALG2**, respectively. An outline of our proof on the approximation ratio is as follows: (I) We first obtain an upper-bound on the number $|ALG2(S)|$ of deleted characters by **ALG2**. Then, (II) we bound the upper-bound above by $\alpha|OPT(S)|$, where $\alpha = (k+2)/2$ if k is even and $\alpha = (k+1)/2$ if k is odd.

(I) We obtain an upper bound on the number $|ALG2(S)|$ of deleted characters by our algorithm **ALG2**. To do so, we first construct a new string \hat{S} , called a “marked” string, by replacing every character $s = \sigma$ in the co-subsequence $\overline{OPT(S)}$ of $OPT(S)$ with an auxiliary symbol $\hat{\sigma} \notin \Sigma$, called a “marked” symbol (character). Let $\hat{\Sigma}$ be the alphabet of marked symbols, where $\hat{\sigma} \in \hat{\Sigma}$ if $\sigma \in \Sigma$. Then, if the i th character $s_i = \sigma$ of S is in $\overline{OPT(S)}$, then it is replaced with the corresponding marked symbol $\hat{\sigma}$. Furthermore, we replace every marked character in \hat{S} with a new symbol $\gamma \notin \Sigma \cup \hat{\Sigma}$, and call it the “ γ -string” S_γ of S . For example, consider $S = a^1b^1a^2b^1c^1a^2b^1c^1b^1c^2b^2$ as an input for MRSD again, where $\Sigma = \{a, b, c\}$. One can verify that $OPT(S) = a^5c^3b^2$ is an optimal solution. Then, we obtain the marked string $\hat{S} = a^1\hat{b}^1a^2\hat{b}^1\hat{c}^1a^2\hat{b}^1c^1\hat{b}^1c^2b^2$, where $\hat{\Sigma} = \{\hat{a}, \hat{b}, \hat{c}\}$ is the alphabet of the marked symbols. By replacing every marked character in \hat{S} with γ , we obtain the γ -string $S_\gamma = a^1\gamma^1a^2\gamma^1\gamma^1a^2\gamma^1c^1\gamma^1c^2b^2 = a^1\gamma^1a^2\gamma^2a^2\gamma^1c^1\gamma^1c^2b^2$. Let r be the number of γ -runs in the γ -string. For example, S_γ includes four γ -runs, i.e., $r = 4$, three length-1 γ -runs and one length-2 γ -run. Note that $r \leq |\overline{OPT(S)}|$ holds.

Next, just for the sake of the analysis on approximation ratios, we introduce a new algorithm **ALG2'** for the γ -string S_γ as input, which is very similar to **ALG2**, but **ALG2'** deletes all the γ -runs. It is important to note that it is not necessary to find any optimal solution. Later, we show that the number $|ALG2'(S_\gamma)|$ of characters deleted by **ALG2'** for S_γ is a good upper bound on $|ALG2(S)|$.

■ **Algorithm 3** A high-level description of our algorithm **ALG2'**.

Input: The γ -string S_γ of S over the alphabet $\Sigma \cup \{\gamma\}$.

Output: A run subsequence $ALG2'(S_\gamma)$ of S_γ .

- 1: For each $\sigma \in \Sigma$, select the leftmost longest σ -run in S_γ .
- 2: Output the concatenation of all the selected longest runs.

► **Example 13.** Consider the γ -string $S_\gamma = a^1\gamma^1a^2\gamma^2a^2\gamma^1c^1\gamma^1c^2b^2$ of S . Then $ALG2'(S_\gamma) = a^2c^2b^2$ is the output of **ALG2'**.

Since all γ -runs are deleted from S_γ , $ALG2'(S_\gamma)$ must be feasible for k -MRSD on the original input S . Then, we get the following upper bound on $|ALG2(S)|$:

► **Lemma 14.** For any input string S and its γ -string S_γ , the following inequalities hold:

1. $|ALG2(S)| \geq |ALG2'(S_\gamma)|$;
2. $|ALG2(S)| \leq |ALG2'(S_\gamma)|$.

Proof. (1) $\ell_{max}(\sigma)$ in the γ -string S_γ is at most $\ell_{max}(\sigma)$ in the original string S for each $\sigma \in \Sigma$. Therefore, $|ALG2(S)| \geq |ALG2'(S_\gamma)|$ holds. (2) From $|S_\gamma| = |S|$ and $|ALG2(S)| \geq |ALG2'(S_\gamma)|$, the number $|ALG2'(S_\gamma)|$ of characters deleted from S_γ is at least the number $|ALG2(S)|$ of deleted characters from S . ◀

(II) Next, we consider an upper bound on the number $|\overline{ALG2'}(S_\gamma)|$ of deleted characters by $ALG2'$ on S_γ . The crux of the following estimation is the number of deleted characters from $OPT(S)$ of S_γ by $ALG2'$.

► **Lemma 15.** *For any input string S and its γ -string S_γ for any optimal solution $OPT(S)$, the following inequality holds:*

$$|\overline{ALG2'}(S_\gamma)| \leq \left(\left\lfloor \frac{k}{2} \right\rfloor + 1 \right) \cdot |\overline{OPT}(S)|$$

Proof. We first divide γ -runs in the γ -string S_γ into two types, **(type-i)** $\sigma_i \gamma^h \sigma_j$ for $\sigma_i \neq \sigma_j$, and **(type-ii)** $\sigma_i \gamma^h \sigma_i$ for an integer $h > 0$. Suppose that S_γ can be represented by $S_\gamma = S_\gamma^1 \circ \gamma^h \circ S_\gamma^2$. Recall that if we delete all γ 's from S_γ , then the remaining sequence must be an optimal solution, i.e., a run subsequence. Therefore, if the middle γ -run is in **(type-i)** and S_γ^1 (resp., S_γ^2) includes a $\sigma \in \Sigma$, then S_γ^2 (resp., S_γ^1) does not include σ . On the other hand, we can see that the second type γ -run partitions some σ_i -run in the optimal solution $OPT(S)$ into the left σ_i -run and the right σ_i -run in the γ -string S_γ .

For example, look at a γ -string $S_\gamma = a^{p_1} \gamma^{h_1} b^{p_2} \gamma^{h_2} b^{p_3} \gamma^{h_3} b^{p_4} \gamma^{h_4} b^{p_5} \gamma^{h_5} c^{p_6} \gamma^{h_6} d^{p_7}$ for $p_1, \dots, p_7 \geq 1$ and $h_1, \dots, h_6 \geq 1$. We focus on the six γ -runs in S_γ . Since the left and the right runs (or characters) of the second (also, the third and the fourth) γ -run in S_γ are the same, it is in **(type-ii)**. Here, we can see that one long b -run of length $(p_2 + p_3 + p_4 + p_5)$ is divided into four b -runs, b^{p_2} , b^{p_3} , b^{p_4} and b^{p_5} . Note that $ALG2'$ selects the longest σ_i -run for each $\sigma_i \in \Sigma$ and deletes all other σ_i -runs from S_γ . Therefore, three b -runs of the four b -runs are deleted by $ALG2'$. If q γ -runs divide a σ -run of length at most k into $q+1$ σ -runs, then the length of each σ -run except for the longest σ -run is bounded above by $\lfloor k/2 \rfloor$. This implies that the number of σ_i 's deleted from $OPT(S)$ for a symbol $\sigma_i \in \Sigma$ is bounded above by $\lfloor k/2 \rfloor$ per γ -run in **(type-ii)** in the worst case. On the other hand, the left and the right runs of the first (also, the fifth and the sixth) γ -run in S_γ are different, and thus it is in **(type-i)**. For example, in the left substring of the fifth length- h_5 γ -run, neither c nor d appears since c and d are included in the right substring of the γ -run. Therefore, after deleting γ^{h_5} , we can independently count the numbers of characters in $OPT(S)$ that are deleted from the left substring and from the right substring by $ALG2'$.

Let the number of γ -runs in **(type-ii)** be r . Note that $r \leq |\overline{OPT}(S)|$ holds and the total number of γ 's in **(type-i)** and **(type-ii)** deleted from S_γ is $|\overline{OPT}(S)|$. Also, the total number of σ 's deleted from $OPT(S)$ for all symbols in Σ is bounded above by $\lfloor k/2 \rfloor \cdot r$. Hence, we obtain the following inequality on the number of characters deleted by $ALG2'$:

$$|\overline{ALG2'}(S_\gamma)| \leq |\overline{OPT}(S)| + \left\lfloor \frac{k}{2} \right\rfloor \cdot r \leq \left(\left\lfloor \frac{k}{2} \right\rfloor + 1 \right) \cdot |\overline{OPT}(S)|$$

This completes the proof of this lemma. ◀

From Lemmas 14 and 15, we obtain the following theorem:

► **Theorem 16.** *$ALG2$ is a linear-time approximation algorithm with approximation ratios $(k+2)/2$ if k is even, and $(k+1)/2$ if k is odd.*

4 Improved approximation algorithm for MRSD

In this section, we present an improved approximation algorithm $ALG3$ that runs in linear time and its approximation ratios are $(k+4)/4$ if k is even, and $(k+3)/4$ if k is odd.

4.1 Concatenation operation

We first define an alternating σ -run:

► **Definition 17.** An alternating σ -run (simply, an alternating run) in $S = S[1, n]$ is a substring $S[i, i + 2p]$ for an integer $p \geq 1$ such that (i) $S[i] = S[i + 2] = \dots = S[i + 2p] = \sigma$, (ii) $S[i + 1] \neq \sigma$, $S[i + 3] \neq \sigma$, \dots , $S[i + 2p - 1] \neq \sigma$, and (iii) $S[i - 2] \neq \sigma$ (if $i - 2 \geq 1$), $S[i - 1] \neq \sigma$ (if $i - 1 \geq 1$), $S[i + 2p + 1] \neq \sigma$ (if $i + 2p + 1 \leq n$), and $S[i + 2p + 2] \neq \sigma$ (if $i + 2p + 2 \leq n$).

For example, consider a string $S = S[1, 15] = ababcbabcbcbcc$. In the string S , $S[1, 3] = aba$, $S[2, 4] = bab$, $S[6, 8] = aba$, $S[7, 13] = babcbcb$, and $S[10, 14] = cbcbcb$ are an alternating a -run, an alternating b -run, an alternating a -run, an alternating b -run, and an alternating c -run, respectively.

Then, we introduce a *concatenation* operation to obtain a σ -run from an alternating σ -run in the string S . Consider an alternating σ -run $S[i, i + 2p]$. Then, the concatenation operation deletes all characters that are not σ from $S[i, i + 2p]$, i.e., $S[i + 1]$, $S[i + 3]$, \dots , $S[i + 2p - 1]$, and obtains a σ -run. For a string $ababa$, however, there are two possibilities, a^3 and b^2 by deleting two b 's and three a 's, respectively. The operation finds a concatenation (i.e., run) as long as possible using an optimal algorithm for the INTERVAL SCHEDULING PROBLEM (see, e.g., [8]): We regard the alternating run $S[i, i + 2p]$ as the interval $[i, i + 2p]$ of *weight* $2p + 1$. A pair of two alternating runs $S[i, i + 2p_i]$ and $S[j, j + 2p_j]$ is independent if $i + 2p_i < j$ or $j + 2p_j < i$ holds. The concatenation operation aims to find a maximum weight subset of mutually independent alternating runs from S , and obtain a long σ -run from the selected alternating σ -run by deleting all the characters that are not σ for every σ .

► **Operation (Concatenation operation).** Given the input string S , the operation obtains a concatenated sequence S_c :

(**Step 1**) Find all the alternating runs in S .

(**Step 2**) Select a maximum subset \mathcal{M} of mutually independent alternating runs in S .

(**Step 3**) Delete all characters from S so that every alternating σ -run in \mathcal{M} becomes a σ -run, and obtain a concatenated sequence S_c .

► **Example 18.** Consider again a string $S = S[1, 15] = ababcbabcbcbcc$. In **Step 1**, we find an alternating a -run $S[1, 3]$ of weight three, an alternating b -run $S[2, 4]$ of weight three, an alternating a -run $S[6, 8]$ of weight three, an alternating b -run $S[7, 13]$ of weight seven, and an alternating c -run $S[10, 14]$ of weight five. Then, in **Step 2**, we select $\mathcal{M} = \{S[1, 3], S[6, 8], S[10, 14]\}$ whose total weight is 11. Finally, we delete four characters $S[2]$, $S[7]$, $S[11]$, $S[13]$ from S , and obtain $S_c = aabcaabccccc = a^2bca^2bc^4$.

We estimate the running time of the concatenation operation. Note that the number of alternating runs overlapped at the same position is at most two, and there are $O(n)$ alternating runs in S . Hence **Step 1** can be implemented in $O(n)$ time by scanning the string S from left to right. Furthermore, during **Step 1**, we can sort the alternating runs according to their right-ends in S . Among $O(n)$ alternating runs, we can select the maximum independent set \mathcal{M} by greedily selecting the non-overlapped alternating run with the leftmost right-end [8] in **Step 2**, which takes $O(n)$ time. **Step 3** needs $O(n)$ time. Therefore, the total running time of the concatenation operation is $O(n)$.

4.2 The algorithm

We present an improved approximation algorithm ALG3 with approximation ratios $(k + 4)/4$ if k is even, and $(k + 3)/4$ if k is odd.

■ **Algorithm 4** A high-level description of our algorithm **ALG3**.

Input: An input string S in which every symbol σ appears at most k times.

Output: A run subsequence $ALG3(S)$ of S .

-
- 1: Execute the concatenation operation and obtain the concatenated sequence S_c of S .
 - 2: For each $\sigma \in \Sigma$, select the leftmost longest σ -run in S_c obtained in the previous step.
 - 3: Output the concatenation of all the selected longest runs.
-

For example, if the concatenation operation produces $S_c = a^2bca^2bc^4$ from S , then **ALG3** outputs $ALG3(S) = a^2bc^4$. It is clear that **ALG3** can obtain a feasible solution in $O(n)$ time.

4.3 Approximation ratios

Let $OPT(S)$ and $ALG3(S)$ be the solutions obtained by an optimal algorithm **OPT**, and **ALG3** for an input string S , respectively. An outline of our proof on the approximation ratio is very similar to Section 3: (I) We first obtain an upper-bound on the number $|ALG3(S)|$ of characters deleted by **ALG3**, comparing the number of characters deleted by the optimal algorithm. Then, (II) we bound the upper-bound above by $\alpha|OPT(S)|$, where $\alpha = (k+4)/4$ if k is even, and $\alpha = (k+3)/4$ if k is odd. Again, we first construct the marked string \hat{S} from S using the optimal algorithm, and design a little bit worse algorithm **ALG3'** than **ALG3**.

Similarly to Section 3, we construct the marked string \hat{S} by replacing every character $s = \sigma$ in the co-subsequence $OPT(S)$ with a symbol $\hat{\sigma} \notin \Sigma$. Let $\hat{\Sigma}$ be the alphabet of marked symbols. For example, consider a string $S = ababcababcbcbcc$ of length 15. Then, $OPT(S) = a^4b^2c^3$ is an optimal solution and $\hat{S} = a\hat{b}a\hat{b}\hat{c}a\hat{b}a\hat{b}\hat{c}b\hat{c}b\hat{c}c$ is the marked string obtained from $OPT(S)$. Note that the optimal solution deletes $S[10] = c$ and thus \hat{S} includes $\hat{S}[10] = \hat{c}$. Furthermore, c does not appear in $\hat{S}[1, 9]$ since $OPT(S)$ must be the run subsequence and $\hat{S}[11, 15]$ includes c 's. In addition, $\hat{S}[1, 6]$ does not include b since b 's appear in $\hat{S}[8, 15]$ and $\hat{S}[7] = \hat{b}$.

Similarly to a σ -run for $\sigma \in \Sigma$, for the marked symbol $\hat{\sigma} \in \hat{\Sigma}$, a $\hat{\sigma}$ -run can be defined. Also, without distinguishing $\hat{\sigma}$ from σ , we consider “mixed” σ -runs, $\sigma^{h_1}\hat{\sigma}^{h_2}$ -type, $\hat{\sigma}^{h_1}\sigma^{h_2}$ -type, and $\hat{\sigma}^{h_1}\sigma^{h_2}\hat{\sigma}^{h_3}$ -type, for some positive integers h_1, h_2 , and h_3 in the following. That is, for example, $\sigma^2\hat{\sigma}^3$ is regarded as one mixed run of length five. Note that we do not need to consider the $\sigma^{h_1}\hat{\sigma}^{h_2}\sigma^{h_3}$ -type since marked strings are constructed based on optimal solutions. Furthermore, we define an alternating-“mixed” run without distinguishing $\hat{\sigma}$ from σ as follows:

► **Definition 19.** An alternating-mixed σ -run (or, simply alternating-mixed run) in $\hat{S} = \hat{S}[1, n]$ is a substring $\hat{S}[i, i+2p]$ which satisfies, (Case 1), (Case 2), or (Case 3):

(Case 1) $\sigma^{h_1}\hat{\sigma}^{h_2}$ -type. (i) $\hat{S}[i] = \hat{S}[i+2] = \dots = \hat{S}[i+2p_1] = \sigma$, and $\hat{S}[i+2(p_1+1)] = \hat{S}[i+2(p_1+2)] = \dots = \hat{S}[i+2p] = \hat{\sigma}$ for $1 \leq p_1 < p$, (ii) $\hat{S}[i+1] \notin \{\sigma, \hat{\sigma}\}$, $\hat{S}[i+3] \notin \{\sigma, \hat{\sigma}\}$, \dots , $\hat{S}[i+2p-1] \notin \{\sigma, \hat{\sigma}\}$, and (iii) $\hat{S}[i-2] \neq \sigma$ (if $i-2 \geq 1$), $\hat{S}[i-1] \neq \sigma$ (if $i-1 \geq 1$), $\hat{S}[i+2p+1] \neq \hat{\sigma}$ (if $i+2p+1 \leq n$), and $\hat{S}[i+2p+2] \neq \hat{\sigma}$ (if $i+2p+2 \leq n$). That is, $\hat{S}[i, i+2p_1]$ and $\hat{S}[i+2(p_1+1), i+2p]$ are the alternating σ -run and the alternating $\hat{\sigma}$ -run, respectively, and $\hat{S}[1, i+2p] = \hat{S}[1, i+2p_1] \circ \sigma' \circ \hat{S}[i+2(p_1+1), i+2p]$ for a symbol $\sigma' \notin \{\sigma, \hat{\sigma}\}$.

(Case 2) $\hat{\sigma}^{h_1}\sigma^{h_2}$ -type. (i) $\hat{S}[i] = \hat{S}[i+2] = \dots = \hat{S}[i+2p_1] = \hat{\sigma}$, and $\hat{S}[i+2(p_1+1)] = \hat{S}[i+2(p_1+2)] = \dots = \hat{S}[i+2p] = \sigma$ for $1 \leq p_1 < p$, (ii) $\hat{S}[i+1] \notin \{\sigma, \hat{\sigma}\}$, $\hat{S}[i+3] \notin \{\sigma, \hat{\sigma}\}$, \dots , $\hat{S}[i+2p-1] \notin \{\sigma, \hat{\sigma}\}$, and (iii) $\hat{S}[i-2] \neq \hat{\sigma}$ (if $i-2 \geq 1$), $\hat{S}[i-1] \neq \hat{\sigma}$ (if $i-1 \geq 1$), $\hat{S}[i+2p+1] \neq \sigma$ (if $i+2p+1 \leq n$), and $\hat{S}[i+2p+2] \neq \sigma$ (if $i+2p+2 \leq n$). That

is, $\hat{S}[i, i + 2p_1]$ and $\hat{S}[i + 2(p_1 + 1), i + 2p]$ are the alternating $\hat{\sigma}$ -run and the alternating σ -run, respectively, and $\hat{S}[1, i + 2p] = \hat{S}[1, i + 2p_1] \circ \sigma' \circ \hat{S}[i + 2(p_1 + 1), i + 2p]$ for a symbol $\sigma' \notin \{\sigma, \hat{\sigma}\}$.

(Case 3) $\hat{\sigma}^{h_1} \sigma^{h_2} \hat{\sigma}^{h_3}$ -type. (i) $\hat{S}[i] = \hat{S}[i + 2] = \dots = \hat{S}[i + 2p_1] = \hat{\sigma}$, $\hat{S}[i + 2(p_1 + 1)] = \hat{S}[i + 2(p_1 + 2)] = \dots = \hat{S}[i + 2p_2] = \sigma$, and $\hat{S}[i + 2(p_2 + 1)] = \hat{S}[i + 2(p_2 + 2)] = \dots = \hat{S}[i + 2p] = \hat{\sigma}$, for $1 \leq p_1 < p_2 < p$, (ii) $\hat{S}[i + 1] \notin \{\sigma, \hat{\sigma}\}$, $\hat{S}[i + 3] \notin \{\sigma, \hat{\sigma}\}$, \dots , $\hat{S}[i + 2p - 1] \notin \{\sigma, \hat{\sigma}\}$, and (iii) $\hat{S}[i - 2] \neq \hat{\sigma}$ (if $i - 2 \geq 1$), $\hat{S}[i - 1] \neq \hat{\sigma}$ (if $i - 1 \geq 1$), $S[i + 2p + 1] \neq \hat{\sigma}$ (if $i + 2p + 1 \leq n$), and $\hat{S}[i + 2p + 2] \neq \hat{\sigma}$ (if $i + 2p + 2 \leq n$). That is, $\hat{S}[i, i + 2p_1]$ and $\hat{S}[i + 2(p_2 + 1), i + 2p]$ are the alternating $\hat{\sigma}$ -runs, and the middle $\hat{S}[i + 2(p_1 + 1), i + 2p_2]$ is the alternating σ -run, and $\hat{S}[1, i + 2p] = \hat{S}[1, i + 2p_1] \circ \sigma' \circ \hat{S}[i + 2(p_1 + 1), i + 2p_2] \circ \sigma'' \circ \hat{S}[i + 2(p_2 + 1), i + 2p]$ for symbols $\sigma', \sigma'' \notin \{\sigma, \hat{\sigma}\}$.

For example, consider the marked string $\hat{S} = \hat{a}\hat{b}\hat{a}\hat{b}\hat{c}\hat{a}\hat{b}\hat{a}\hat{b}\hat{c}\hat{b}\hat{c}\hat{c}$ of length 15. In the string \hat{S} , $\hat{S}[2, 4] = \hat{b}\hat{a}\hat{b}$, $\hat{S}[7, 13] = \hat{b}\hat{a}\hat{b}\hat{c}\hat{b}\hat{c}$, and $\hat{S}[10, 14] = \hat{c}\hat{b}\hat{c}\hat{b}\hat{c}$ are the alternating b -run, the alternating-mixed b -run in **(Case 3)**, and the alternating-mixed c -run in **(Case 2)**, respectively.

Here, we introduce the concatenation operation for marked strings by slightly modifying the concatenation operation.

► **Operation** (Concatenation operation for marked strings). Given the marked string \hat{S} of the input string S , the operation obtains a concatenated sequence \hat{S}_c :

(Step 1) Find all the alternating runs and all the alternating-mixed runs on $\Sigma \cup \hat{\Sigma}$ in S .

(Step 2) Select a maximum subset \mathcal{M} of mutually independent alternating/alternating-mixed runs in S .

(Step 3) Delete all characters that are neither σ nor $\hat{\sigma}$ from S so that every alternating σ -run, every alternating $\hat{\sigma}$ -run and every alternating-mixed σ -run in \mathcal{M} become a σ -run, a $\hat{\sigma}$ -run and a mixed σ -run, respectively, and obtain the concatenated sequence \hat{S}_c .

► **Example 20.** Consider the marked string $\hat{S} = \hat{a}\hat{b}\hat{a}\hat{b}\hat{c}\hat{a}\hat{b}\hat{a}\hat{b}\hat{c}\hat{b}\hat{c}\hat{c}$ of length 15. In **Step 1**, we find an alternating a -run $S[1, 3]$ of weight three, an alternating \hat{b} -run $S[2, 4]$ of weight three, an alternating a -run $S[6, 8]$ of weight three, an alternating-mixed b -run $S[7, 13]$ of weight seven, and an alternating-mixed c -run $S[10, 14]$ of weight five. Then, in **Step 2**, we select $\mathcal{M} = \{S[1, 3], S[6, 8], S[10, 14]\}$ whose total weight is 11. Finally, we delete four characters $S[2]$, $S[7]$, $S[11]$, and $S[13]$ from \hat{S} , and obtain $\hat{S}_c = \hat{a}\hat{a}\hat{b}\hat{c}\hat{a}\hat{a}\hat{b}\hat{c}\hat{c}\hat{c} = \hat{a}^2\hat{b}\hat{c}\hat{a}^2\hat{b}\hat{c}\hat{c}^3$.

To obtain an upper bound on $|\overline{ALG3}(S)|$, we introduce the following algorithm **ALG3'**:

■ **Algorithm 5** A high-level description of our algorithm **ALG3'**.

Input: The marked string \hat{S} of S over the alphabet $\Sigma \cup \hat{\Sigma}$.

Output: A run subsequence $ALG3'(\hat{S})$ of \hat{S} .

-
- 1: Execute the concatenation operation for marked strings and obtain the concatenated sequence \hat{S}_c of \hat{S} .
 - 2: For each mixed σ -run in \hat{S}_c , replace every $\hat{\sigma}$ with σ .
 - 3: For each $\sigma \in \Sigma$, select the leftmost longest σ -run in the sequence obtained in the previous step.
 - 4: Output the concatenation of all the selected longest runs.
-

Note that the replacement $\hat{\sigma}$ in the second step of **ALG3'** does not create a new σ -run, but only increases the length of the σ -run which originally appears in \hat{S}_c .

► **Example 21.** If the concatenated sequence is $\hat{S}_c = a^2\hat{b}\hat{c}a^2b\hat{c}c^3$, then we replace \hat{c} in the rightmost mixed run, obtain $a^2\hat{b}\hat{c}a^2bc^4$, and finally output a^2bc^4 of length seven. Recall that an optimal solution is $OPT(S) = a^4b^2c^3$ of length nine.

Using arguments very similar to the proof of Lemma 14, we obtain the following lemma:

► **Lemma 22.** *For any input S and its marked string \hat{S} , the following inequalities hold:*

1. $|ALG3(S)| \geq |ALG3'(\hat{S})|;$
2. $\overline{ALG3(S)} \leq \overline{ALG3'(\hat{S})}.$

Recall that the optimal algorithm **OPT** deletes the number $|\overline{OPT(S)}|$ of characters from the input string S . In the following, we show that, given the marked string \hat{S} of S , the number $|ALG3'(\hat{S})|$ of characters deleted by the worse algorithm **ALG3'** from \hat{S} is bounded above by $\frac{k+4}{4}|\overline{OPT(S)}|$ if k is even, and $\frac{k+3}{4}|\overline{OPT(S)}|$ if k is odd.

We first consider the case $k = 2$:

► **Lemma 23.** *Suppose that $k = 2$. Then, for S and its marked string \hat{S} , the following inequality holds:*

$$\overline{ALG3'(\hat{S})} \leq \frac{3}{2}|\overline{OPT(S)}|.$$

Proof. We investigate the concatenation operation for marked strings. Consider a marked symbol $\hat{\sigma}_i$. Assume that $\sigma_i \neq \sigma_j$. Suppose that the marked string \hat{S} includes a substring $\sigma_j\hat{\sigma}_i\sigma_j$. Then, even if the middle character $\hat{\sigma}_i$ is deleted in the concatenation operation for marked strings, this deletion of $\hat{\sigma}_i$ does not increase the number of deleted characters compared to the number of deleted characters by the optimal algorithm. Therefore, we assume that $\hat{\sigma}_i$ remains in the concatenated sequence \hat{S}_c after the concatenation operation for marked strings. Note that \hat{S} does not include a substring $\sigma_j\hat{\sigma}_i\sigma_j\hat{\sigma}_i$. The reason is as follows. The rightmost $\hat{\sigma}_i$ implies that the optimal algorithm deletes the rightmost σ_i , but if it is not deleted, then the length of the optimal solution increases by one, which is a contradiction. One sees that it is enough to count the deleted characters in $OPT(S)$ independently within substrings of length three or four in \hat{S} , since here we assume that $k = 2$.

- Suppose that \hat{S} includes a substring $\hat{\sigma}_i\sigma_j\hat{\sigma}_i$. If the middle σ_j is deleted, then $\hat{\sigma}_i^2$ is obtained by the concatenation operation for marked strings. **ALG3'** deletes those two $\hat{\sigma}_i$'s in the third step. In other words, **ALG3'** deletes one character in $OPT(S)$ per two marked characters in $\overline{OPT(S)}$ (i.e., $1/2$ in $OPT(S)$ per one in $\overline{OPT(S)}$).
- Suppose that \hat{S} includes a substring $\sigma_j\hat{\sigma}_i\sigma_j\sigma_i$. If the right σ_j is deleted, then we obtain $\sigma_j\hat{\sigma}_i\sigma_i$ and the middle $\hat{\sigma}_i$ is replaced by σ_i in the second step of **ALG3'**. Here, we can see that **ALG3'(\hat{S})** misses one σ_j but adds one σ_i , i.e., the number of deleted characters remains the same for the substring.

In summary, **ALG3'** deletes one character in $OPT(S)$ for two marked characters in $\overline{OPT(S)}$. That is, **ALG3'** deletes possibly all the marked characters in $\overline{OPT(S)}$, and in addition, at most $\frac{1}{2}|\overline{OPT(S)}|$ nonmarked characters in total. From these considerations, the upper bound on the number of deleted characters of **ALG3'** can be calculated as follows:

$$\overline{ALG3'(\hat{S})} \leq |\overline{OPT(S)}| + \frac{1}{2}|\overline{OPT(S)}| = \frac{3}{2}|\overline{OPT(S)}|. \quad (10)$$

◀

Next, we obtain the following lemma for $k = 3$, whose proof appears in Appendix B.

► **Lemma 24.** *Suppose that $k = 3$. Then, for S and its marked string \hat{S} , the following inequality holds:*

$$|\overline{ALG3'}(\hat{S})| \leq \frac{3}{2} |\overline{OPT}(S)|.$$

Finally, we show the case $k \geq 4$:

► **Lemma 25.** *Suppose that $k \geq 4$. Then, for S and its marked string \hat{S} , the following inequality holds:*

$$|\overline{ALG3'}(\hat{S})| \leq \frac{\lfloor \frac{k}{2} \rfloor + 2}{2} |\overline{OPT}(S)|.$$

Proof. We count the deleted characters that appear in $OPT(S)$, during the concatenation operation for marked strings. Note that if \hat{S} includes a substring consisting of at least two consecutive marked characters, then the substring may partition some σ -run in $OPT(S)$ into the left σ -run and the right σ -run in \hat{S} , and the length of the shorter σ -run is at most $\lfloor k/2 \rfloor$.

Now we observe the marked string \hat{S} that includes a substring $\sigma \hat{\sigma} b \hat{\sigma} b \hat{\sigma} b \cdots \hat{\sigma} b \hat{\sigma} \sigma'$. Suppose that $\sigma = b$ and $\sigma' = b$ in the substring. Then, all $\hat{\sigma}$'s are deleted in the concatenation operation for marked strings since the alternating b -run is longer than the alternating $\hat{\sigma}$ -run. Here, the number of deleted characters by the concatenation operation remains the same as that by the optimal algorithm. If $\sigma \neq b$ or $\sigma' \neq b$, then all b 's in $\hat{\sigma} b \hat{\sigma} b \hat{\sigma} b \cdots \hat{\sigma} b \hat{\sigma}$ can be deleted by the concatenation operation for marked strings and the $\hat{\sigma}$ -run is produced, i.e., characters in $OPT(S)$ can be additionally deleted by $ALG3'$.

- Suppose that the length of the produced $\hat{\sigma}$ -run by the concatenation operation for marked strings is p . This implies that $p - 1$ b 's that appear in the optimal solution are deleted while p $\hat{\sigma}$'s remain in the concatenated sequence \hat{S}_c obtained in the first step of $ALG3'$.
- Suppose that after the first step of $ALG3'$, we obtain the concatenated sequence \hat{S}_c . Moreover, suppose that \hat{S}_c includes the $\hat{\sigma}$ -run of length at least two. Then, \hat{S}_c possibly includes a substring $\sigma_0^{p_1} \hat{\sigma}^p \sigma_0^{p_2}$ for $p \geq 2$. Namely, we can see that the σ_0 -run of length $p_1 + p_2$ is divided into two runs of length p_1 and p_2 by the $\hat{\sigma}$ -run of length $p \geq 2$. Recall that in the third step $ALG3'$ selects the longest σ_0 -run. That is, if $p_1 \leq p_2$, then p_1 characters are deleted; otherwise, p_2 characters are deleted by $ALG3'$. From $p_1 + p_2 \leq k$, at most $\lfloor k/2 \rfloor$ characters are deleted if the length of the $\hat{\sigma}$ -run is at least two.

In summary, the number of characters deleted from $OPT(S)$ in \hat{S} is one for every one single $\hat{\sigma}_1^1$, and $\lfloor k/2 \rfloor$ for every one substring of consecutive marked characters $\hat{\sigma}_1 \cdots \hat{\sigma}_p$ for $p \geq 2$.

Suppose that \hat{S}_c includes r_1 single marked characters whose left and right characters are not marked, and r_2 substrings of at least two consecutive marked characters $\hat{\sigma}_1 \cdots \hat{\sigma}_p$. Note that $r_1 + 2r_2 \leq |\overline{OPT}(S)|$, and now we assume that $k \geq 4$. Hence, we obtain the following inequality on the number of characters deleted by $ALG3'$ from the marked string \hat{S} :

$$\begin{aligned} |\overline{ALG3'}(\hat{S})| &\leq |\overline{OPT}(S)| + r_1 + \left\lfloor \frac{k}{2} \right\rfloor \cdot r_2 \leq |\overline{OPT}(S)| + \frac{1}{2} \cdot \left\lfloor \frac{k}{2} \right\rfloor (r_1 + 2r_2) \\ &\leq \frac{\lfloor \frac{k}{2} \rfloor + 2}{2} \cdot |\overline{OPT}(S)| \end{aligned}$$

This completes the proof. ◀

From Lemmas 22, 23, 24 and 25, we obtain the following theorem:

► **Theorem 2.** *k -MRS can be approximated in linear time within a factor of $(k + 4)/4$ for even k , and $(k + 3)/4$ for odd k .*

Finally, we can show that the analyses on the approximation ratios of $(k + 4)/4$ and $(k + 3)/4$ are strictly tight. For details, the reader is referred to Appendix D.

References

- 1 Amit Agarwal, Moses Charikar, Konstantin Makarychev, and Yury Makarychev. $O(\sqrt{\log n})$ approximation algorithms for min UnCut, min 2CNF deletion, and directed cut problems. In Harold N. Gabow and Ronald Fagin, editors, *Proceedings of the 37th Annual ACM Symposium on Theory of Computing, Baltimore, MD, USA, May 22-24, 2005*, pages 573–581. ACM, 2005. doi:10.1145/1060590.1060675.
- 2 Yuichi Asahiro, Hiroshi Eto, Mingyang Gong, Jesper Jansson, Guohui Lin, Eiji Miyano, Hirotaka Ono, and Shunichi Tanaka. Approximation algorithms for the longest run subsequence problem. In Laurent Bulteau and Zsuzsanna Lipták, editors, *34th Annual Symposium on Combinatorial Pattern Matching, CPM 2023, June 26-28, 2023, Marne-la-Vallée, France*, volume 259 of *LIPICs*, pages 2:1–2:12. Schloss Dagstuhl – Leibniz-Zentrum für Informatik, 2023. doi:10.4230/LIPICs.CPM.2023.2.
- 3 Bengt Aspvall, Michalel F. Plass, and Robert E. Tarjan. A linear-time algorithm for testing the truth of certain quantified boolean formulas. *Information Processing Letters*, 8(3):121–123, 1979. doi:10.1016/0020-0190(79)90002-4.
- 4 Irit Dinur and Shmuel Safra. The importance of being biased. In John H. Reif, editor, *Proceedings on 34th Annual ACM Symposium on Theory of Computing, May 19-21, 2002, Montréal, Québec, Canada*, pages 33–42. ACM, 2002. doi:10.1145/509907.509915.
- 5 Riccardo Dondi and Florian Sikora. The longest run subsequence problem: Further complexity results. In Pawel Gawrychowski and Tatiana Starikovskaya, editors, *32nd Annual Symposium on Combinatorial Pattern Matching, CPM 2021, July 5-7, 2021, Wrocław, Poland*, volume 191 of *LIPICs*, pages 14:1–14:15. Schloss Dagstuhl – Leibniz-Zentrum für Informatik, 2021. doi:10.4230/LIPICs.CPM.2021.14.
- 6 Håstad Johan. Some optimal inapproximability results. *Journal of the ACM*, 48(4):498–859, 2001.
- 7 George Karakostas. A better approximation ratio for the vertex cover problem. In Luís Caires, Giuseppe F. Italiano, Luís Monteiro, Catuscia Palamidessi, and Moti Yung, editors, *Automata, Languages and Programming, 32nd International Colloquium, ICALP 2005, Lisbon, Portugal, July 11-15, 2005, Proceedings*, volume 3580 of *Lecture Notes in Computer Science*, pages 1043–1050. Springer, 2005. doi:10.1007/11523468_84.
- 8 Jon Kleinberg and Éva Tardos. *Algorithm Design*. Addison Wesley, 2006.
- 9 Michael Lewin, Dror Livnat, and Uri Zwick. Improved rounding techniques for the MAX 2-SAT and MAX DI-CUT problems. In William J. Cook and Andreas S. Schulz, editors, *Integer Programming and Combinatorial Optimization, 9th International IPCO Conference, Cambridge, MA, USA, May 27-29, 2002, Proceedings*, volume 2337 of *Lecture Notes in Computer Science*, pages 67–82. Springer, 2002. doi:10.1007/3-540-47867-1_6.
- 10 Junwei Luo, Yawei Wei, Mengna Lyu, Zhengjiang Wu, Xiaoyan Liu, Huimin Luo, and Chaokun Yan. A comprehensive review of scaffolding methods in genome assembly. *Briefings Bioinform.*, 22(5), 2021. doi:10.1093/bib/bbab033.
- 11 Sven Schrinner, Manish Goel, Michael Wulfert, Philipp Spohr, Korbinian Schneeberger, and Gunnar W. Klau. The longest run subsequence problem. In Carl Kingsford and Nadia Pisanti, editors, *20th International Workshop on Algorithms in Bioinformatics, WABI 2020, September 7-9, 2020, Pisa, Italy (Virtual Conference)*, volume 172 of *LIPICs*, pages 6:1–6:13. Schloss Dagstuhl – Leibniz-Zentrum für Informatik, 2020. doi:10.4230/LIPICs.WABI.2020.6.
- 12 Sven Schrinner, Manish Goel, Michael Wulfert, Philipp Spohr, Korbinian Schneeberger, and Gunnar W. Klau. Using the longest run subsequence problem within homology-based scaffolding. *Algorithms Mol. Biol.*, 16(1):11, 2021. doi:10.1186/s13015-021-00191-8.
- 13 David Zuckerman. Linear degree extractors and the inapproximability of max clique and chromatic number. *Theory Comput.*, 3(1):103–128, 2007. doi:10.4086/TOC.2007.V003A006.

A

 Proof of Lemma 11

► **Lemma 11.** *The summation of y_i, z_i satisfies the following inequality:*

$$\sum_{i=0}^k y_i \leq n_0 + n_1 + \sum_{i=0}^k z_i.$$

Proof. Note that $\sum_{i=0}^k y_i = |\Lambda'_3|$ and $\sum_{i=0}^k z_i = |\Lambda''_3|$. Moreover, $n_0 + n_1 = |\Lambda_1|$. Therefore it is sufficient to show $|\Lambda'_3| \leq |\Lambda_1| + |\Lambda''_3| = |\Lambda_1 \cup \Lambda''_3|$.

Then we design a mapping from Λ'_3 to $\Lambda_1 \cup \Lambda''_3$ as follows: Consider a symbol $s \in \Lambda'_3$. Since s is a bad symbol, the algorithm chooses an s -run in a bad segment sts with $t \neq s$. Then we know that t in this bad segment sts must be in $ALG1(S')$. Otherwise, the local operation-1 is applicable to $ALG1(S')$, which contradicts Step 4 of **ALG1**. If t is unique, then we map s to t , which is called the *Case-1 mapping*. Now, we can assume t is not unique.

Since t is in $ALG1(S')$ and t is not unique, then t must be in $\Lambda_2 \cup \Lambda_3$ since, otherwise, if t is in Π , then a t -run of length two must be chosen in $ALG1(S')$. If t is in Λ''_3 , then we map s to t , which is called the *Case-2 mapping*. Otherwise, $t \in \Lambda_2 \cup \Lambda'_3$ and thus the length of t in $ALG1(S')$ is exactly one. Note that t is not unique, and the local operation-2 is not applicable to $ALG1(S')$. So, there exists another t such that it is in a bad segment wtw where a w -run of length two is chosen by the algorithm. Therefore $w \in \Lambda''_3$ and we map s to w , which is called the *Case-3 mapping*.

Then we prove that the mapping is injective and we are done. Suppose s_1, s_2 are two distinct symbols in Λ'_3 such that they are mapping to the same symbol $w \in \Lambda_1 \cup \Lambda''_3$. If w is an unique symbol, then s_1, s_2 are mapping to w by the Case-1 mapping. That is, s_1ws_1 and s_2ws_2 are two substrings of S' , which is impossible since $s_1 \neq s_2$ and w is unique. So, we can assume w is not unique, i.e., $w \in \Lambda''_3$ and s_1, s_2 are mapping to w by the Case-2 mapping or the Case-3 mapping.

The first case is that both s_1, s_2 are mapping to w by the Case-2 mapping. Then it indicates s_1ws_1 and s_2ws_2 are two bad segments in S' where w is in Λ''_3 . However, this case is impossible since two w -runs in $ALG1(S')$ must come from a bad segment associated with w .

The second case is that exactly one of s_1, s_2 maps to w by the Case-2 mapping and the other one maps to w by the Case-3 mapping. Without loss of generality, we assume that s_1 and s_2 map to w by the Case-2 and Case-3 mapping, respectively. Recall that in the bad segment s_1ws_1 , the symbol w is in Λ''_3 . So, there is a substring ws_1ws_1 or s_1ws_1w containing two bad segments in S' such that the w -run of length two is in $ALG1(S')$. By symmetry, we consider the substring ws_1ws_1 . Since s_2 maps to w by Case-3 mapping, $s_2s_1s_2$ is the substring such that an s_1 and an s_2 are in $ALG1(S')$, which leads to a contradiction since $s_1 \in \Lambda'_3$.

The remaining case is that both s_1, s_2 map to w by the Case-3 mapping. In this case, there exist two substrings s_1ws_1 and s_2ws_2 in S' and $w \in \Lambda''_3$. However, this is impossible since two w -runs must come from a bad segment of w , which completes the proof. ◀

B

 Proof of Lemma 24

► **Lemma 24.** *Suppose that $k = 3$. Then, for S and its marked string \hat{S} , the following inequality holds:*

$$|\overline{ALG3'(\hat{S})}| \leq \frac{3}{2} |\overline{OPT(S)}|.$$

Proof. Consider a marked symbol $\hat{\sigma}_i$ and assume that $\sigma_i \neq \sigma_j$ again. As before, we consider the case where $\hat{\sigma}_i$ remains in the concatenated sequence \hat{S}_c after the concatenation operation for marked strings. Note that \hat{S} does not include a substring $\sigma_j \hat{\sigma}_i \sigma_j \hat{\sigma}_i \sigma_j \hat{\sigma}_i$. The reason is the same as the previous: The rightmost $\hat{\sigma}_i$ implies that the optimal algorithm deletes the rightmost σ_i , but if the rightmost σ_i is not deleted, then the length of the optimal solution increases by one, which is a contradiction.

- Suppose that \hat{S} includes a substring $\hat{\sigma}_i \sigma_j \hat{\sigma}_i$. If the middle σ_j is deleted, then $\hat{\sigma}_i^2$ is obtained by the concatenation operation for marked strings, and **ALG3'** deletes two characters $\hat{\sigma}_i^2$ in the third step. In other words, **ALG3'** deletes one character in $OPT(S)$ per two marked characters in $OPT(S)$.
- Suppose that \hat{S} includes a substring $\sigma_j \hat{\sigma}_i \sigma_j \sigma_i$. If the right σ_j is deleted, then we obtain $\sigma_j \hat{\sigma}_i \sigma_i$ and the middle $\hat{\sigma}_i$ is replaced by σ_i in the second step of **ALG3'**. Namely, $ALG3'(\hat{S})$ misses one σ_j but adds one σ_i , i.e., the number of deleted characters remains the same for the substring.
- Suppose that \hat{S} includes a substring $\sigma_j \hat{\sigma}_i \sigma_j \hat{\sigma}_i \sigma_j \sigma_i$. If the right two σ_j 's are deleted, then we obtain $\sigma_j \hat{\sigma}_i^2 \sigma_i$ and two $\hat{\sigma}_i^2$ are replaced by σ_i^2 in the second step of **ALG3'**. Namely, $ALG3'(\hat{S})$ misses two σ_j 's but adds two σ_i 's, i.e., the number of deleted characters remains the same for the substring.

Again, we conclude that **ALG3'** deletes at least one character in $OPT(S)$ for two marked characters in $OPT(S)$. From these considerations, the upper bound on the number of deleted characters of **ALG3'** can be calculated as follows:

$$|\overline{ALG3'(\hat{S})}| \leq |\overline{OPT(S)}| + \frac{1}{2} |\overline{OPT(S)}| = \frac{3}{2} |\overline{OPT(S)}|. \quad (11)$$

◀

C Bad examples for Theorem 1

We can provide tight examples for the analysis on the approximation ratios when $k = 2$ and $k = 3$ in Theorem 1.

- (i) First, suppose that $k = 2$. Then, consider the following string $S^2 = S_1^2 \circ S_2^2 \circ \dots \circ S_p^2$ of length $6p$ for some integer p :

$$S^2 = \underbrace{\text{substring } S_1^2}_{abcabc} \underbrace{\text{substring } S_2^2}_{degdeg} \cdots \underbrace{\text{substring } S_p^2}_{\sigma_{3p-2}\sigma_{3p-1}\sigma_{3p}\sigma_{3p-2}\sigma_{3p-1}\sigma_{3p}}.$$

Then, an optimal solution is $OPT(S^2) = a^2bcd^2eg \cdots \sigma_{3p-2}^2\sigma_{3p-1}\sigma_{3p}$ of length $4p$, and a solution of **ALG1** is $ALG1(S^2) = abcdeg \cdots \sigma_{3p-2}\sigma_{3p-1}\sigma_{3p}$ of length $3p$. Hence, $|\overline{ALG1(S^2)}|/|\overline{OPT(S^2)}| = 4/3$.

- (ii) Next, suppose that $k = 3$. Then, the following string $S^3 = S_1^3 \circ S_2^3 \circ \dots \circ S_p^3$ of length $9p$ for some integer p is a tight example:

$$S^3 = \underbrace{\text{substring } S_1^3}_{abcabcabc} \underbrace{\text{substring } S_2^3}_{degdegdeg} \cdots \underbrace{\text{substring } S_p^3}_{\sigma_{3p-2}\sigma_{3p-1}\sigma_{3p}\sigma_{3p-2}\sigma_{3p-1}\sigma_{3p}\sigma_{3p-2}\sigma_{3p-1}\sigma_{3p}}.$$

Then, an optimal solution is $OPT(S^3) = a^3bcd^3eg \cdots \sigma_{3p-2}^3\sigma_{3p-1}\sigma_{3p}$ of length $5p$, and an solution of **ALG1** is the same as $ALG1(S^3)$. Therefore, $|\overline{ALG1(S^3)}|/|\overline{OPT(S^3)}| = 5/3$.

- (iii) For the case $k \geq 4$, the approximation-ratio analysis in Theorem 1 is almost tight, but there is a slight gap. Suppose that $k \geq 4$. Then, consider the following string $S^k = S_1^k \circ S_2^k \circ \dots \circ S_{2p}^k$ of length $6p$ for some integer $3k$:

$$S^k = \underbrace{\text{substring } S_1^k \text{ of length } 3k/2}_{aabaab \dots aab} \underbrace{\text{substring } S_2^k \text{ of length } 3k/2}_{ccbcb \dots ccb} \dots$$

$$\underbrace{\text{substring } S_{2p-1}^k \text{ of length } 3k/2}_{\sigma_{3p-2}\sigma_{3p-2}\sigma_{3p-1}\sigma_{3p-2}\sigma_{3p-2}\sigma_{3p-1} \dots \sigma_{3p-2}\sigma_{3p-2}\sigma_{3p-1}}$$

$$\underbrace{\text{substring } S_{2p}^k \text{ of length } 3k/2}_{\sigma_{3p}\sigma_{3p}\sigma_{3p-1}\sigma_{3p}\sigma_{3p}\sigma_{3p-1} \dots \sigma_{3p}\sigma_{3p}\sigma_{3p-1}}$$

Then, the following run subsequence $OPT(S^k)$ of length $(2k+1)p$ is an optimal solution:

$$OPT(S^k) = a^k c^k b d^k g^k e \dots \sigma_{3p-2}^k \sigma_{3p}^k \sigma_{3p-1}^k$$

On the other hand, our algorithm ALG2 outputs the following subsequence of length $5p$:

$$ALG2(S^k) = a^2 b c^2 d^2 e g^2 \dots \sigma_{2p-2}^2 \sigma_{3p-1}^2 \sigma_{3p}^2.$$

Hence, $|\overline{ALG2(S^k)}|/|\overline{OPT(S^k)}| = (2k+1)/5$.

D Bad examples for Theorem 2

We can show that the analysis on the approximation ratios in Theorem 2 is tight.

- (i) First, suppose that k is even. Then, consider the following string $S^e = S_1^e S_2^e \dots S_p^e$ of length $2pk$ for some integer p :

$$S^e = \underbrace{\text{substring } S_1^e}_{a^{\frac{k}{2}} b^2 a^{\frac{k}{2}} b^{k-2}} \underbrace{\text{substring } S_2^e}_{c^{\frac{k}{2}} d^2 c^{\frac{k}{2}} d^{k-2}} \dots \underbrace{\text{substring } S_p^e}_{\sigma_{2p-1}^{\frac{k}{2}} \sigma_{2p}^2 \sigma_{2p-1}^{\frac{k}{2}} \sigma_{2p}^{k-2}}$$

Then, the optimal solution $OPT(S^e)$ is obtained by deleting $2p$ characters as follows:

$$OPT(S^e) = a^k b^{k-2} c^k d^{k-2} \dots \sigma_{2p-1}^k \sigma_{2p}^{k-2}$$

On the other hand, our algorithm ALG3 selects the leftmost longest σ_i -run for each $i \in \{1, \dots, 2p\}$ and thus $ALG3(S^e)$ is as follows:

$$ALG3(S^e) = a^{\frac{k}{2}} b^{k-2} c^{\frac{k}{2}} d^{k-2} \dots \sigma_{2p-1}^{\frac{k}{2}} \sigma_{2p}^{k-2}$$

The total number of characters deleted from S^e is $\frac{k+4}{2} \cdot p$. Hence, $|\overline{ALG3(S^e)}|/|\overline{OPT(S^e)}| = (k+4)/4$.

- (ii) Next, suppose that k is odd. Then, consider the following string $S^o = S_1^o S_2^o \dots S_p^o$ of length $2pk$ for some integer p :

$$S^o = \underbrace{\text{substring } S_1^o}_{a^{\frac{k+1}{2}} b^2 a^{\frac{k-1}{2}} b^{k-2}} \underbrace{\text{substring } S_2^o}_{c^{\frac{k+1}{2}} d^2 c^{\frac{k-1}{2}} d^{k-2}} \dots \underbrace{\text{substring } S_p^o}_{\sigma_{2p-1}^{\frac{k+1}{2}} \sigma_{2p}^2 \sigma_{2p-1}^{\frac{k-1}{2}} \sigma_{2p}^{k-2}}$$

Very similarly, we obtain the following equality: $|\overline{ALG3(S^o)}|/|\overline{OPT(S^o)}| = (k+3)/4$. As a result, the analysis of the approximation ratios in the proof of Theorem 2 is tight.