Approximation Algorithms for the Longest Run Subsequence Problem

Yuichi Asahiro
Kyushu Sangyo University, Fukuoka, Japan

Hiroshi Eto
Kyushu Institute of Technology, Iizuka, Japan

Mingyang Gong
Uniersity of Alberta, Edmonton, Canada

Jesper Jansson
Kyoto University, Kyoto, Japan

Guohui Lin
Uniersity of Alberta, Edmonton, Canada

Eiji Miyano
Kyushu Institute of Technology, Iizuka, Japan

Hirotaka Ono
Nagoya University, Nagoya, Japan

Shunichi Tanaka
Kyushu Institute of Technology, Iizuka, Japan

Abstract

We study the approximability of the Longest Run Subsequence problem (LRS for short). For a string $S = s_1 \cdots s_n$ over an alphabet Σ, 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 sequence in which every symbol $\sigma \in \Sigma$ occurs in at most one run. Given a string S, the goal of LRS is to find a longest run subsequence S^* of S such that the length $|S^*|$ is maximized over all the run subsequences of S. It is known that LRS is APX-hard even if each symbol has at most two occurrences in the input string, and that LRS admits a polynomial-time k-approximation algorithm if the number of occurrences of every symbol in the input string is bounded by k. In this paper, we design a polynomial-time $\frac{k+1}{2}$-approximation algorithm for LRS under the k-occurrence constraint on input strings. For the case $k = 2$, we further improve the approximation ratio from $\frac{3}{2}$ to $\frac{4}{3}$.

2012 ACM Subject Classification Theory of computation → Design and analysis of algorithms

Keywords and phrases Longest run subsequence problem, bounded occurrence, approximation algorithm

Digital Object Identifier 10.4230/LIPIcs.CPM.2023.2

Funding This work is partially supported by NSERC Canada, and JSPS KAKENHI Grant Numbers JP17K00024, JP20H05967, JP21K11755, JP21K19765, JP22H00513, JP22H03550, and JP22K11915.

Acknowledgements The authors would like to thank the anonymous reviewers for their suggestions and detailed comments that helped to improve the presentation of the paper.

1 Introduction

The main goal of genome analysis is to study and compare genetic content among organisms, and thus genome sequencing to determine the complete sequence of a genome is one of its most important stages. Since the first whole genome was obtained [10], genome sequencing technologies have significantly improved. Almost all the current DNA sequencing technologies are based on the following process: First, tens or hundreds of millions of fragments from random positions on the DNA sequence are read via shotgun sequencing. Second, these randomly extracted fragments, called reads, are merged to form a set of contiguous sequences, called contigs, by using an assembly algorithm. Then, the contigs are ordered correctly in a phase called *scaffolding*. One commonly used approach for scaffolding is to rearrange contigs by comparing two or more incomplete assemblies of related samples (see, for example, [8]).

34th Annual Symposium on Combinatorial Pattern Matching (CPM 2023).
Editors: Laurent Bulteau and Zsuzsanna Lipták; Article No. 2; pp. 2:1–2:12
Leibniz International Proceedings in Informatics
Schloss Dagstuhl – Leibniz-Zentrum für Informatik, Dagstuhl Publishing, Germany

In the context of the scaffolding phase of genome assembly, the One-Sided Scaffold Filling problem [9], Two-Sided Scaffold Filling problem [7], One-Side-Filled Longest Common Subsequence problem [3], and Two-Side-Filled Longest Common Subsequence problem [4] were formulated as combinatorial optimization problems on two strings. For those problems, their computational complexities were proved, and then fixed-parameter tractable algorithms, approximation algorithms, and exponential-time exact algorithms were proposed in [2, 3, 4, 7]. Very recently, as a different formulation of the scaffolding phase, Schrinner et al. [11, 12] introduced the Longest Run Subsequence problem (LRS for short), defined as follows: For a string $S = s_1 \cdots s_n$ over an alphabet Σ, 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 sequence in which every symbol $\sigma \in \Sigma$ occurs in at most one run. Given a string S, the goal of LRS is to find a longest run subsequence S^* of S such that the length $|S^*|$ is maximized over all the run subsequences of S.

▶ **Example 1.** Consider the string $S = abacacbbab$ over the alphabet $\Sigma = \{a, b, c\}$. It contains (i) four runs of symbol a, i.e., a in the first position, a in the third position, a in the fifth position, and a in the ninth position, (ii) three runs of symbol b, i.e., b in the second position, bb in the seventh and eighth positions, and b in the tenth position, and (iii) two runs of c, i.e., c in the fourth position, and c in the sixth position in S. The numbers of occurrences of a, b, and c are four, four, and two, respectively.

An optimal solution to LRS on input S is $S^* = aaccbbb$. For example, the leftmost run aa of length two in S^* is obtained from the leftmost substring aba in S by deleting the second character b. One sees that S^* is a run subsequence, i.e., S^* contains (at most) one run for every symbol. The length of S^* is seven. Note that $S' = aaacbbb$ is another optimal solution since $|S'|$ is also seven. ⌟

Schrinner et al. [12] showed that LRS is NP-hard. Subsequently, Dondi and Sikora [5] showed that LRS is APX-hard even if each symbol has at most two occurrences in the input string, and that LRS admits a polynomial-time $\min\{|\Sigma|, k\}$-approximation algorithm if the number of occurrences of every symbol in the input string is bounded by k.

In this paper, we propose the following improved approximation algorithms for LRS:

- We first design a polynomial-time $\frac{k+1}{2}$-approximation algorithm for LRS, when the number of occurrences of every symbol is at most k.
- For the case $k = 2$, we further improve the approximation ratio from $\frac{3}{2}$ to $\frac{4}{3}$.

Related work. The fixed-parameter tractability and the parameterized complexity of LRS have been previously investigated [5, 12]: Schrinner et al. [12] showed that there is an $O(|\Sigma| \cdot |S| \cdot 2^{|\Sigma|})$-time algorithm, given a string S over an alphabet Σ as input of LRS, i.e., LRS is fixed-parameter tractable when parameterized by the size $|\Sigma|$ of the alphabet on which the input string is defined. Dondi and Sikora [5] showed that LRS can be solved by a randomized algorithm in $O(2^r \cdot r \cdot |S|^3)$ time and polynomial space, where r is the number of different runs in a solution, and thus $r \leq |S|$. They also proved that LRS admits a polynomial kernel when parameterized by the length of the solution, but that it does not admit a polynomial kernel when parameterized by the size $|\Sigma|$ of the alphabet or by the number r of runs.

2 Preliminaries

Let Σ be a finite alphabet of symbols. 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., $|S| = n$. A *subsequence* of S is a sequence $s_{i_1} \cdots s_{i_m}$, such that $1 \leq i_1 < i_2 < \cdots < i_m \leq |S|$. Let $S[i]$ denote the character of S in the ith position for $1 \leq i \leq |S|$, and $S[i, j]$ denote the substring of S that starts from the ith position and ends at the jth position. For a symbol σ, we denote by σ^k a string that is the concatenation of k occurrences of symbol σ for some integer $k \geq 1$. A *run* in S is a substring $S[i, j]$ such that: (1) $S[i] = S[i + 1] = \cdots = S[j]$; (2) $S[i - 1] \neq S[i]$ if $i > 1$; and (3) $S[j + 1] \neq S[j]$ if $j < |S|$. For any $\sigma \in \Sigma$, a run in S of the form σ^k is called a *length-k σ-run* in S. Observe that if $S[i, j]$ is a σ-run, then it has length $j - i + 1$. Given a string S on alphabet Σ, a *run subsequence S' of S* is a subsequence in which every symbol $\sigma \in \Sigma$ occurs in at most one run.

Let $occ(\sigma)$ be the number of occurrences of σ in the input string S. Let $occ_{max}(S) = \max_{\sigma \in S} occ(\sigma)$. For example, consider a string $S = abacaabbab$. Then, S includes four a-runs, a, a, a^2, and a, three b-runs, b, b^2, and b, and one length-1 c-run. The number $occ(a)$ of occurrences of a is five. Also, $occ(b) = 4$ and $occ(c) = 1$. Therefore, $occ_{max}(S) = 5$.

Our problem LRS can be formulated as follows:

▶ **Problem 2** (Longest Run Subsequence problem, LRS). *Given an alphabet Σ and a string $S = s_1 \cdots s_n$ with $s_i \in \Sigma$, 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.*

Schrinner et al. [12] show that LRS is NP-hard by giving a polynomial-time reduction from the Linear Ordering problem, which is shown to be NP-hard in [6]. In this paper we consider the following restricted LRS:

▶ **Problem 3** (k-Longest Run Subsequence problem, k-LRS). *If the maximum number $occ_{max}(S)$ of occurrences of symbols in the input 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. Dondi and Sikora [5] show that 2-LRS remains hard even from the approximation point of view; they give an L-reduction from the Minimum Independent Set on Cubic Graph problem, which is shown to be APX-hard in [1]:

▶ **Proposition 4** ([5]). *2-LRS is APX-hard.*

Suppose that an input string of k-LRS is S over an alphabet Σ. Also, without loss of generality, we assume here that every symbol in Σ appears at least once, and the maximum number $occ_{max}(S)$ of occurrences of symbols in S is k. Note that the length of an optimal run subsequence is bounded by $k|\Sigma|$. Consider the following two simple algorithms, (i) and (ii):

(i) Arbitrarily select one run of every symbol $\sigma \in \Sigma$ in S, and construct a run subsequence S' by concatenating all the selected runs.

One sees that $|S'|$ is at least $|\Sigma|$. Therefore, we can conclude that k-LRS is k-approximable.

(ii) Find a symbol, say, σ of the maximum occurrences k, and construct another run subsequence $S'' = \sigma^k$.

Then, we can conclude that k-LRS is $|\Sigma|$-approximable. By using those two algorithms, we obtain the following proposition:

▶ **Proposition 5** ([5]). *There is a $\min(|\Sigma|, k)$-approximation algorithm for k-LRS.*

Since $\min(|\Sigma|, k) \leq \sqrt{|S|}$, the above proposition implies the following:

▶ **Corollary 6** ([5]). *The general LRS problem admits a $\sqrt{|S|}$-approximation algorithm.*

3 A polynomial-time $\frac{k+1}{2}$-approximation algorithm for k-LRS

In this section, we improve the approximation ratio for k-LRS from k to $\frac{k+1}{2}$. Our approximation algorithm ALG uses a very natural idea:

Algorithm ALG. Given an input string S over an alphabet Σ, ALG selects a longest σ-run in S for each $\sigma \in \Sigma$, and outputs the concatenation of all the selected longest runs.

▶ **Example 7.** Consider the input string $S = abacaabbab$ (for 5-LRS). The longest a-run, b-run, c-run are aa in the fifth and sixth positions, bb in the seventh and eighth positions, and c in the fourth position. Therefore, the output of ALG is $ALG = caabb$. ⌟

We now prove that the above simple algorithm achieves the claimed approximability bound:

▶ **Theorem 8.** *ALG is a polynomial-time $\frac{k+1}{2}$-approximation algorithm for k-LRS.*

Proof. Clearly, ALG returns a valid solution since one run is selected for every symbol in S, and runs in polynomial time. We bound its approximation ratio in the following. Let S be an input string of k-LRS. We assume that S consists of m symbols, i.e., $|\Sigma| = m$, and $occ_{max}(S) = k$. Then, suppose that OPT and ALG are solutions obtained by an optimal algorithm and our algorithm ALG, respectively, for the input S. We consider the following two cases: **(Case 1)** The length of every run in S is one, and **(Case 2)** the length of some run in S is at least two.

(Case 1). Suppose that the length of every run in S is one. Let m_ℓ be the number of symbols in OPT such that the length of the run of those symbols is exactly ℓ $(\leq k)$.

First, the following two equalities hold:

$$|OPT| = \sum_{i=1}^{k} i \cdot m_i; \text{ and} \tag{1}$$

$$|ALG| = m. \tag{2}$$

Let D be the number of characters deleted from S by the optimal algorithm. Since $|\Sigma| = \sum_{i=0}^{k} m_i = m$ and $occ_{max}(S) = k$, the following is satisfied:

$$|OPT| = |S| - D \leq km - D. \tag{3}$$

We now derive a lower bound on D. Suppose that a symbol σ_2 in S appears exactly twice in the optimal solution OPT, i.e., OPT contains the length-2 σ_2-run $\sigma_2\sigma_2$. Recall that the length of all the runs in the input string S is one. Namely, there is at least one different character, say, σ' between two σ_2's in S. That is, σ' must be deleted from S in order to obtain the length-2 σ_2-run. Since OPT contains m_2 symbols such that the length of the runs of those symbols is exactly two, the total number of deleted characters from S to obtain the length-2 runs is at least m_2. It is important to note that the character-deletion to obtain each run is independently carried out, and therefore the number of deleted characters is not doubly counted. Similarly, the total number of deleted characters from S to obtain the length-ℓ runs is at least $(\ell - 1)m_\ell$ for each $3 \leq \ell \leq k$. As a result, we obtain the following lower bound on D:

$$D \geq m_2 + 2m_3 + \cdots + (k-1)m_k = \sum_{i=2}^{k}(i-1)m_i = \sum_{i=1}^{k}(i-1)m_i. \tag{4}$$

From Eq.(3) and Eq.(4), the following inequality holds:

$$|OPT| \leq km - \sum_{i=1}^{k}(i-1)m_i.$$

From Eq.(1), this can be rewritten as:

$$|OPT| \leq (k+1)m - |OPT|,$$

and then rearranged to give:

$$|OPT| \leq \frac{(k+1)m}{2}.$$

From Eq.(2), we obtain the following approximation ratio:

$$\frac{|OPT|}{|ALG|} \leq \frac{k+1}{2}.$$

(Case 2). Suppose that the length of a σ-run in S is at least two and S consists of symbols in Σ. For every such symbol $\sigma \in \Sigma$, we consider a different symbol $\overline{\sigma}$, called a *dummy* symbol. Then, we insert $\overline{\sigma}$ between every consecutive two symbols $\sigma\sigma$ in S so that the two σ's are not consecutive. Hence we obtain a longer sequence S_d such that the length of all the runs in S_d is one. For example, consider a string

$$S = abacaabbbab.$$

Then, we insert a dummy $\overline{a}$ between the fifth and the sixth positions, a dummy $\overline{b}$ between the seventh and the eighth positions, and the other dummy $\overline{b}$ between the eighth and the ninth positions as follows:

$$S_d = abaca\overline{a}ab\overline{b}b\overline{b}bab.$$

Note that the number of occurrences of each dummy $\overline{\sigma}$ is at most $k-1$ since the maximum number $occ_{max}(S)$ of occurrences of (original) symbols in S is bounded by k. Suppose that OPT_d and ALG_d are solutions obtained by an optimal algorithm and our algorithm `ALG`, respectively, for the input S_d. One sees that the maximum number $occ_{max}(S_d)$ of occurrences of symbols in S_d is also bounded by k. Therefore, from the arguments in **(Case 1)**, the following inequality is satisfied:

$$\frac{|OPT_d|}{|ALG_d|} \leq \frac{k+1}{2}. \tag{5}$$

The original input S is a subsequence of S_d. Hence, the following clearly holds:

$$|OPT| \leq |OPT_d|. \tag{6}$$

Now consider ALG and ALG_d. (i) For each symbol σ such that the length of all the σ-runs is one, its dummy $\overline{\sigma}$ is not inserted into S_d. Hence, ALG and ALG_d contain one σ, but, of course, neither contains any $\overline{\sigma}$. (ii) If the maximum length of a σ-run in S is (at least) two for some symbol σ, then ALG contains (at least) two σ's. On the other hand, ALG_d contains one σ and one dummy $\overline{\sigma}$ instead. From (i) and (ii), we have:

$$|ALG| \geq |ALG_d|. \tag{7}$$

From the three inequalities (5), (6), and (7), the following approximation ratio is obtained again:

$$\frac{|OPT|}{|ALG|} \le \frac{|OPT_d|}{|ALG_d|} \le \frac{k+1}{2}.$$

For both cases **(Case 1)** and **(Case 2)**, the approximation ratio of `ALG` is bounded above by $\frac{k+1}{2}$. ◀

▶ Remark 9. To see that the approximation analysis above is tight, consider the following string S, where $|S| = n = 2k\ell$, and $\sigma_i \neq \sigma_j$ for $i \neq j$.

$$S = \overbrace{\sigma_1\sigma_2\sigma_1\sigma_2\cdots\sigma_1\sigma_2}^{2k}\overbrace{\sigma_3\sigma_4\sigma_3\sigma_4\cdots\sigma_3\sigma_4}^{2k}\ \cdots\ \overbrace{\sigma_{2\ell-1}\sigma_{2\ell}\sigma_{2\ell-1}\sigma_{2\ell}\cdots\sigma_{2\ell-1}\sigma_{2\ell}}^{2k}.$$

Namely, the length-$2k$ prefix string contains k σ_1's and k σ_2's alternatively. The next string of length $2k$ contains k σ_3's and k σ_4's alternatively, and so on. Then, we can obtain the following run subsequence S' by deleting $k-1$ σ_2's from the first length-$2k$ prefix string, $k-1$ σ_4's from the next string of length $2k$, and so on:

$$S' = \sigma_1^k\sigma_2\sigma_3^k\sigma_4\ \cdots\ \sigma_{2\ell-1}^k\sigma_{2\ell}.$$

Hence, the length of OPT is at least $|S'| = (k+1)\ell$. On the other hand, the solution ALG of our algorithm `ALG` for S contains one of the k σ_i's for each $1 \le i \le 2\ell$:

$$ALG = \sigma_1\sigma_2\cdots\sigma_{2\ell}.$$

The length of ALG is 2ℓ. As a result,

$$\frac{|OPT|}{|ALG|} \ge \frac{k+1}{2}.$$

This shows that the analysis of the approximation ratio in the proof of Theorem 8 is tight. ⌟

Recall that we can always return a run subsequence of length k as shown in the previous section, and k-LRS is $|\Sigma|$-approximable. Therefore, we obtain the following corollary:

▶ **Corollary 10.** *There is a polynomial-time* $\min\{|\Sigma|, \frac{k+1}{2}\}$*-approximation algorithm for* k*-LRS.*

4 A polynomial-time $\frac{4}{3}$-approximation algorithm for 2-LRS

For 2-LRS, `ALG` achieves the approximation ratio of $\frac{3}{2}$. In this section we improve the approximation ratio to $\frac{4}{3}$.

As shown in Remark 9, the following string S is a bad example for `ALG`.

$$S = ababcdcdefef.$$

One sees that from the leftmost substring $S[1,4] = abab$ of length four (resp. $S[5,8] = cdcd$ and $S[9,12] = efef$), we can only obtain a run subsequence of length at most three, i.e., the length of any optimal solution is at most nine. Therefore, one of the possible optimal solution OPT for S is:

$$OPT = aabccdeef.$$

The solution ALG of $\texttt{ALG}$ for S is:

$$ALG = abcdef.$$

Namely, OPT has two a's (resp. two c's and two e's), but $\texttt{ALG}$ has only one a (resp. one c and one e). This observation suggests to us that if there is only one character, say, σ' between two occurrences of a symbol σ, then we should delete σ' and obtain a run $\sigma\sigma$ of length two. This is a basic strategy of our new algorithm $\texttt{ALG}_2$.

Before describing details of $\texttt{ALG}_2$, we give some definitions which are used in the following. Let S be an input string. Assume that all the symbols in Σ appear in S. We define several subsets of Σ in the following.

- Let $\Sigma_1 = \{\sigma \mid occ(\sigma) = 1, \sigma \in \Sigma\}$ be a set of symbols that appear exactly once in the input string S.
- Let $\Sigma_2 = \{\sigma \mid occ(\sigma) = 2, \sigma \in \Sigma\}$ be a set of symbols that appear exactly twice in the input string S.

Note that $\Sigma = \Sigma_1 \cup \Sigma_2$ in 2-LRS. Now, we consider a symbol $\sigma \in \Sigma_2$ and define several disjoint subsets of Σ_2. In the following, by *distance* we mean the number of characters between the two occurrences of a symbol.

- If two σ's consecutively appear in S, then we call σ a distance-0 symbol. Let $\Sigma_{2,0}$ be a subset of all the distance-0 symbols in Σ_2.
- If there is one character between two σ's, then we call σ a distance-1 symbol. Let $\Sigma_{2,1}$ be a subset of all the distance-1 symbols in Σ_2.
- We define $\Sigma_{2,\geq 2} = \Sigma_2 \setminus (\Sigma_{2,0} \cup \Sigma_{2,1})$, i.e., for each $\sigma \in \Sigma_{2,\geq 2}$, σ appears twice in S and there are at least two characters between the two σ's.

Next, consider a symbol $\gamma \in \Sigma_1$. As a special case, the left and the right symbols of γ can be the same symbol $\gamma' \in \Sigma_{2,1}$, i.e., the input string S possibly contains a substring $\gamma'\gamma\gamma'$ of length 3, called a *special triple*.

- Let Γ_1 be a set of center symbols of special triples. Note that $\Gamma_1 \subseteq \Sigma_1$.
- Let $\Gamma_{2,1}$ be a set of left and right symbols of special triples. Note that $\Gamma_{2,1} \subseteq \Sigma_{2,1}$.

One sees that $|\Gamma_1| = |\Gamma_{2,1}|$.

Finally, consider two symbols σ and σ' in $\Sigma_{2,1} \setminus \Gamma_{2,1}$ in the input string S such that the substring(s) containing σ and σ' can be represented by (i) $S = \cdots\sigma\sigma'\sigma\sigma'\cdots$, or (ii) $S = \cdots\sigma\lambda\sigma\cdots\sigma'\lambda'\sigma'\cdots$, where both λ and λ' are in $\Sigma_{2,\geq 2}$. (i) If S contains $\sigma\sigma'\sigma\sigma'$ as a substring, then we say that a pair of σ and σ' is called a Ψ-pair. Then, σ and σ' belong to a set $\Psi_{2,1}$. (ii) If $\lambda = \lambda'$, then we say that a pair of σ and σ' is a Λ-pair related to λ. Then, σ and σ' belong to a set $\Lambda_{2,1}$ and λ belongs to $\Lambda_{2,\geq 2}$. Note that $|\Lambda_{2,1}| = 2|\Lambda_{2,\geq 2}|$.

Algorithm. The following is a description of our algorithm $\texttt{ALG}_2$. During execution of $\texttt{ALG}_2$, we determine which characters are included into the run subsequence ALG_2 or not, step by step. Finally, $\texttt{ALG}_2$ outputs the concatenation of the characters (or the subsequences) included into ALG_2 in each step.

Algorithm $\texttt{ALG}_2$.

Input An input string S over an alphabet Σ such that every symbol in Σ appears at most twice.

Output A run subsequence.

Step 1. Count the number of occurrences of every symbol in Σ, and divide Σ to two subsets Σ_1 and Σ_2. Then, examine the distance of every symbol in Σ_2, and obtain $\Sigma_{2,0}$, $\Sigma_{2,1}$, and $\Sigma_{2,\geq 2}$.

Step 2. Find all the special triples, all the Ψ-pairs, and all the Λ-pairs.

Step 3. For every $\sigma \in \Sigma_{2,0}$, the length-2 σ-run σ^2 is included into ALG_2.

Step 4. For every $\sigma \in \Sigma_{2,1}$, execute the following:

- **(i)** For every special triple $\gamma'\gamma\gamma'$, the first two characters $\gamma' \in \Gamma_{2,1}$ and $\gamma \in \Gamma_1$ are included into ALG_2. That is, the third character γ' of that special triple is not included into ALG_2.
- **(ii)** For every Ψ-pair of σ and σ', i.e., for each string $\sigma\sigma'\sigma\sigma'$, its subsequence $\sigma\sigma'\sigma'$ is included into ALG_2. That is, the third character σ of that string is not included into ALG_2.
- **(iii)** For every Λ-pair related to λ of σ and σ', i.e., for two strings $\sigma\lambda\sigma$ and $\sigma'\lambda\sigma'$, two subsequences $\sigma\lambda$, and σ'^2 are included into ALG_2. That is, the third character σ of the former string and the second character λ of the latter string are not included into ALG_2.
- **(iv)** For every $\sigma \in \Sigma_{2,1} \setminus (\Gamma_{2,1} \cup \Psi_{2,1} \cup \Lambda_{2,1})$, σ^2 is included into ALG_2. That is, the character between the two σ's is not included into ALG_2.

Step 5. For every $\sigma \in \Sigma_{2,\geq 2} \setminus \Lambda_{2,\geq 2}$, only the first occurrence of σ is included into ALG_2. That is, if neither of the two occurrences of σ is determined whether or not to be included into ALG_2, then the first occurrence is included into ALG_2 and the other not into ALG_2[1]. If only one occurrence remains undetermined, then it is included into ALG_2.

Step 6. Every $\sigma \in \Sigma_1 \setminus \Gamma_1$ is included into ALG_2.

Step 7. Output the concatenation of the characters and the subsequences included into ALG_2 in Step 3 through Step 6 as a run subsequence, and then halt.

▶ Remark 11. Importantly, the output run subsequence of $\mathtt{ALG}_2$ includes at least one occurrence of every symbol in Σ. ⌟

▶ **Example 12.** To clarify the behavior of $\mathtt{ALG}_2$, we take a look at the following input string of length 20:

$$S = abacdbdecefgfhhijkjk.$$

One sees that $\Sigma_1 = \{g, i\}$, $\Sigma_{2,0} = \{h\}$, $\Sigma_{2,1} = \{a, d, e, f, j, k\}$, and $\Sigma_{2,\geq 2} = \{b, c\}$. (Step 3) $S[14, 15] = hh$ is included into ALG_2. (Step 4-(i)) Since $f \in \Sigma_{2,1}$ and $g \in \Sigma_1$, $S[10, 12] = fgf$ is a special triple. Therefore, we select fg from fgf. (Step 4-(ii)) Since there is a substring $S[17, 20] = jkjk$, the pair of j and k is a Ψ-pair, $\Psi_{2,1} = \{j, k\}$. Then, jkk is included into ALG_2. (Step 4-(iii)) S contains $S[1, 3] = aba$ and $S[5, 7] = dbd$ and thus the pair of a and d is a Λ-pair related to b; $\Lambda_{2,1} = \{a, d\}$ and $\Lambda_{2,\geq 2} = \{b\}$. Hence, ab and dd are included into ALG_2. (Step 4-(iv)) From $S[8, 10] = ece$, we obtain a run e^2 of length two, and $S[9] = c$ is not included into ALG_2. (Step 5) The fourth character c is included into ALG_2 since $c \in \Sigma_{2,\geq 2} \setminus \Gamma_{2,\geq 2}$ and $S[9] = c$ is not included into ALG_2 in Step 4-(iv). (Step 6) The

[1] Alternatively, we can choose any one of the two occurrences of each symbol, to obtain the same approximation ratio.

16th character i is included into ALG_2 since $i \in \Sigma_1 \setminus \Gamma_1$. (Step 7) Finally, the following concatenation of the characters and the subsequences obtained in Step 3 through Step 6 is output as the run subsequence ALG_2 of length 15:

$$ALG_2 = abcddeefghhijkk.$$

⌟

▶ **Theorem 13.** *$\mathtt{ALG}_2$ is a polynomial-time $\frac{4}{3}$-approximation algorithm for 2-LRS.*

Proof. Clearly, $\mathtt{ALG}_2$ returns a valid solution and runs in polynomial time. We bound its approximation ratio in the following. Suppose that OPT and ALG_2 are run subsequences obtained by an optimal algorithm and our algorithm $\mathtt{ALG}_2$, respectively, for the input string S.

We assume that the optimal run subsequence OPT consists of the following symbols (OPT1 through OPT4) or characters in special triples (OPT5):

(OPT1) Consider symbols in $\Sigma_{2,\geq 2}$. Suppose that there are $m_{2,\geq 2,2}$ symbols such that two occurrences of each of them are included into OPT by deleting all the characters between two occurrences. Also, suppose that there are $m_{2,\geq 2,1}$ (resp. $m_{2,\geq 2,0}$) symbols such that one occurrence (resp. no occurrence) of each of them is included into OPT.

(OPT2) Consider symbols in $\Sigma_{2,1} \setminus \Gamma_{2,1}$. Suppose that there are $m_{2,1,2}$ symbols such that two occurrences of each of them are included into OPT by deleting one character between two occurrences. Also, suppose that there are $m_{2,1,1}$ (resp. $m_{2,1,0}$) symbols such that one occurrence (resp. no occurrence) of each of them is included into OPT.

(OPT3) Consider symbols in $\Sigma_{2,0}$. Suppose that there are $m_{2,0,2}$ (resp. $m_{2,0,0}$) symbols such that two occurrences (resp. no occurrence) of each of them are included into OPT. Remark that since the goal is to maximize the length of the run subsequence, we can assume that two occurrences (one run of length two) of the symbol in $\Sigma_{2,0}$ are completely included into OPT, or completely deleted.

(OPT4) Consider symbols in $\Sigma_1 \setminus \Gamma_1$. Suppose that there are $m_{1,1}$ (resp. $m_{1,0}$) symbols such that one occurrence (resp. no occurrence) of each of them is included into OPT.

(OPT5) Consider special triples. For example, take a look at $\gamma'\gamma\gamma'$ where $\gamma \in \Gamma_1$ and $\gamma' \in \Gamma_{2,1}$. One sees that we cannot select all the three characters into any solution subsequence since it can contain at most one run for every symbol. Therefore, OPT includes at most two characters of the special triple, γ'^2, $\gamma'\gamma$, or $\gamma\gamma'$. Since the goal is to maximize the length of the run subsequence, we can assume that OPT includes one of the two characters of the special triple, or does not include any character from the special triple. Suppose that there are $m_{\gamma,2}$ (resp. $m_{\gamma,0}$) special triples such that two characters (resp. no character) of each of them are included into OPT.

Then, the length of OPT is calculated as follows:

$$|OPT| = \overbrace{2m_{2,\geq 2,2} + m_{2,\geq 2,1}}^{\mathsf{OPT1}} + \overbrace{2m_{2,1,2} + m_{2,1,1}}^{\mathsf{OPT2}} + \overbrace{2m_{2,0,2}}^{\mathsf{OPT3}} + \overbrace{m_{1,1}}^{\mathsf{OPT4}} + \overbrace{2m_{\gamma,2}}^{\mathsf{OPT5}}. \tag{8}$$

Now, let D be the number of deleted symbols from S by the optimal algorithm. Then, D is counted by the above assumption:

$$D = \overbrace{m_{2,\geq 2,1} + 2m_{2,\geq 2,0}}^{\mathsf{OPT1}} + \overbrace{m_{2,1,1} + 2m_{2,1,0}}^{\mathsf{OPT2}} + \overbrace{2m_{2,0,0}}^{\mathsf{OPT3}} + \overbrace{m_{1,0}}^{\mathsf{OPT4}} + \overbrace{m_{\gamma,2} + 3m_{\gamma,0}}^{\mathsf{OPT5}}. \tag{9}$$

Next, we consider a lower bound on D. As for symbols in $\Sigma_{2,\geq 2}$, we assumed in (OPT1) that there are $m_{2,\geq 2,2}$ symbols such that two occurrences of each of them are included into OPT, i.e., at least two characters between the two occurrences must be deleted. Also, as for symbols in $\Sigma_{2,1} \setminus \Gamma_{2,1}$, we assumed in (OPT2) that there are $m_{2,1,2}$ symbols such that two occurrences of each of them are included into OPT, i.e., one character between the two occurrences must be deleted. As a result, the following inequality holds:

$$D \geq 2m_{2,\geq 2,2} + m_{2,1,2}. \tag{10}$$

Now, we estimate the length of the output run subsequence of $\mathtt{ALG}_2$.

(ALG1) Consider symbols in $\Sigma_{2,0}$. In Step 3, two occurrences of every symbol in $\Sigma_{2,0}$ are included into ALG_2, i.e., $2m_{2,0,2} + 2m_{2,0,0}$ characters are included into ALG_2.

(ALG2) Consider symbols in $\Gamma_{2,1}$. In Step 4-(i), one occurrence of every symbol in $\Gamma_{2,1}$ is included into ALG_2, i.e., $m_{\gamma,2} + m_{\gamma,0}$ characters are totally included in ALG_2.

(ALG3) Consider symbols in Σ_1. In Step 4-(i), every symbol in Γ_1 ($\subseteq \Sigma_1$) is included into ALG_2. In Step 6, every symbol in $\Sigma_1 \setminus \Gamma_1$ is included into ALG_2. That is, all the symbols in Σ_1 are included into ALG_2. In total, $m_{1,1} + m_{1,0} + m_{\gamma,2} + m_{\gamma,0}$ characters are included into ALG_2.

(ALG4) Consider symbols in $\Sigma_{2,\geq 2}$. In Step 4-(iii), one occurrence of every symbol in $\Lambda_{2,\geq 2}$ ($\subseteq \Sigma_{2,\geq 2}$) is included into ALG_2. Also, in Step 5, one occurrence of every symbol in $\Sigma_{2,\geq 2} \setminus \Lambda_{2,\geq 2}$ is included into ALG_2. In total, $m_{2,\geq 2,2} + m_{2,\geq 2,1} + m_{2,\geq 2,0}$ characters are included into ALG_2.

(ALG5) Consider symbols in $\Sigma_{2,1} \setminus \Gamma_{2,1}$. Recall that $|\Sigma_{2,1} \setminus \Gamma_{2,1}| = m_{2,1,2} + m_{2,1,1} + m_{2,1,0}$. Consider a Ψ-pair of σ and σ', i.e., a substring $\sigma\sigma'\sigma\sigma'$ of length four in S. In Step 4-(ii), three characters σ, σ', and σ' are selected from the Ψ-pair of σ and σ'. Namely, we can see that three characters per two symbols are included into ALG_2. Also, in Step 4-(iii), three characters σ, σ', and σ' are selected from every Λ-pair of σ and σ'. Again, three characters per two symbols are included into ALG_2. In Step 4-(iv), two occurrences of every symbol in $(\Sigma_{2,1} \setminus \Gamma_{2,1}) \setminus (\Psi_{2,1} \cup \Lambda_{2,1})$ are included into ALG_2. As a result, at least $\frac{3}{2}(m_{2,1,2} + m_{2,1,1} + m_{2,1,0})$ characters are included into ALG_2.

Then, the following inequality on the length of ALG_2 holds:

$$\begin{aligned}|ALG_2| \geq\ & \overbrace{m_{2,\geq 2,2} + m_{2,\geq 2,1} + m_{2,\geq 2,0}}^{\text{ALG4}} + \overbrace{\frac{3}{2}(m_{2,1,2} + m_{2,1,1} + m_{2,1,0})}^{\text{ALG5}} \\ & + \overbrace{2m_{2,0,2} + 2m_{2,0,0}}^{\text{ALG1}} + \overbrace{m_{1,1} + m_{1,0} + 2m_{\gamma,2} + 2m_{\gamma,0}}^{\text{ALG2 and ALG3}}.\end{aligned} \tag{11}$$

From Eq.(9) and Eq.(10), we obtain the following inequality:

$$\begin{aligned}& \frac{1}{3}(m_{2,\geq 2,1} + 2m_{2,\geq 2,0} - m_{2,1,2} + m_{2,1,1} + 2m_{2,1,0} + 2m_{2,0,0} + m_{1,0} + m_{\gamma,2} + 3m_{\gamma,0}) \\ \geq\ & \frac{2}{3}m_{2,\geq 2,2}.\end{aligned} \tag{12}$$

Therefore, from Eq.(8) and Eq.(12), $|OPT|$ is bounded as follows:

$$\begin{aligned}|OPT| &= \left(\frac{4}{3}m_{2,\geq 2,2} + \frac{2}{3}m_{2,\geq 2,2}\right) + m_{2,\geq 2,1} + 2m_{2,1,2} + m_{2,1,1} \\ &\quad + 2m_{2,0,2} + m_{1,1} + 2m_{\gamma,2} \\ &\leq \frac{4}{3}m_{2,\geq 2,2} + \frac{4}{3}m_{2,\geq 2,1} + \frac{2}{3}m_{2,\geq 2,0} + \frac{5}{3}m_{2,1,2} + \frac{4}{3}m_{2,1,1} + \frac{2}{3}m_{2,1,0} \\ &\quad + 2m_{2,0,2} + \frac{2}{3}m_{2,0,0} + m_{1,1} + \frac{1}{3}m_{1,0} + \frac{7}{3}m_{\gamma,2} + m_{\gamma,0}\end{aligned} \tag{13}$$

One can verify that the following is satisfied from Eq.(11) and Eq.(13):

$$\frac{|OPT|}{|ALG_2|} \leq \frac{4}{3}.$$ ◀

▶ Remark 14. Again, we can show the tightness for the approximation ratio $\frac{4}{3}$ of $\texttt{ALG}_2$. Consider the following string S, where $|S| = n = 6\ell$.

$$S = \sigma_1\sigma_2\sigma_3\sigma_1\sigma_2\sigma_3\sigma_4\sigma_5\sigma_6\sigma_4\sigma_5\sigma_6 \ \cdots \ \sigma_{3\ell-2}\sigma_{3\ell-1}\sigma_{3\ell}\sigma_{3\ell-2}\sigma_{3\ell-1}\sigma_{3\ell}.$$

Then, we can find the following run subsequence S':

$$S' = \sigma_1^2\sigma_2\sigma_3\sigma_4^2\sigma_5\sigma_6 \ \cdots \ \sigma_{3\ell-2}^2\sigma_{3\ell-1}\sigma_{3\ell}$$

Therefore, the length of OPT is at least $|S'| = 4\ell$. On the other hand, the solution of our algorithm $\texttt{ALG}_2$ for S contains only one of the two σ_i's for each $1 \leq i \leq 3\ell$ since every symbol is in $\Sigma_{2,\geq 2}$:

$$ALG_2 = \sigma_1\sigma_2 \cdots \sigma_{3\ell}.$$

The length of ALG_2 is 3ℓ. As a result,

$$\frac{|OPT|}{|ALG_2|} \geq \frac{4}{3}.$$

This shows that the above approximation analysis is tight. ⌟

5 Conclusion

We have presented a polynomial-time $\frac{k+1}{2}$-approximation algorithm for k-LRS, where the number of occurrences of every symbol in the input string is at most k. Then, for the case $k = 2$, we have reduced the approximation ratio to $\frac{4}{3}$. The current approximation algorithm for 2-LRS is a little bit complicated, and thus might be simplified to obtain the same approximation ratio. Future work is to further improve the approximation ratio of $\frac{4}{3}$ for 2-LRS, and to design an even better approximation algorithm for general k-LRS. It would also be useful to derive tight bounds on the polynomial-time approximation hardness of k-LRS in terms of k.

References

1 Paola Alimonti and Viggo Kann. Some APX-completeness results for cubic graphs. *Theor. Comput. Sci.*, 237(1-2):123–134, 2000. doi:10.1016/S0304-3975(98)00158-3.

2 Yuichi Asahiro, Jesper Jansson, Guohui Lin, Eiji Miyano, Hirotaka Ono, and Tadatoshi Utashima. Polynomial-time equivalences and refined algorithms for longest common subsequence variants. In Hideo Bannai and Jan Holub, editors, *33rd Annual Symposium on Combinatorial Pattern Matching, CPM 2022, June 27-29, 2022, Prague, Czech Republic*, volume 223 of *LIPIcs*, pages 15:1–15:17. Schloss Dagstuhl - Leibniz-Zentrum für Informatik, 2022. doi:10.4230/LIPIcs.CPM.2022.15.

3 Mauro Castelli, Riccardo Dondi, Giancarlo Mauri, and Italo Zoppis. The longest filled common subsequence problem. In Juha Kärkkäinen, Jakub Radoszewski, and Wojciech Rytter, editors, *28th Annual Symposium on Combinatorial Pattern Matching, CPM 2017, July 4-6, 2017, Warsaw, Poland*, volume 78 of *LIPIcs*, pages 14:1–14:13. Schloss Dagstuhl - Leibniz-Zentrum für Informatik, 2017. doi:10.4230/LIPIcs.CPM.2017.14.

4 Mauro Castelli, Riccardo Dondi, Giancarlo Mauri, and Italo Zoppis. Comparing incomplete sequences via longest common subsequence. *Theor. Comput. Sci.*, 796:272–285, 2019. doi:10.1016/j.tcs.2019.09.022.

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 Martin Grötschel, Michael Jünger, and Gerhard Reinelt. A cutting plane algorithm for the linear ordering problem. *Oper. Res.*, 32(6):1195–1220, 1984. doi:10.1287/opre.32.6.1195.

7 Haitao Jiang, Chunfang Zheng, David Sankoff, and Binhai Zhu. Scaffold filling under the breakpoint and related distances. *IEEE ACM Trans. Comput. Biol. Bioinform.*, 9(4):1220–1229, 2012. doi:10.1109/TCBB.2012.57.

8 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.

9 Adriana Muñoz, Chunfang Zheng, Qian Zhu, Victor A. Albert, Steve Rounsley, and David Sankoff. Scaffold filling, contig fusion and comparative gene order inference. *BMC Bioinform.*, 11:304, 2010. doi:10.1186/1471-2105-11-304.

10 F. Sanger, G.M. Air, B.G. Barrell, N.L. Brown, A.R. Coulson, J.C. Fiddes, C.A. Hutchison III, P.M. Slocombe, and M. Smith. Nucleotide sequence of bacteriophage ϕx174 DNA. *Nature*, 265:687–695, 1977.

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.