A 2-Approximation Algorithm for the Complementary Maximal Strip Recovery Problem

Haitao Jiang
Department of Computer Science and Technology, Shandong University, China
htjiang@sdu.edu.cn

Jiong Guo
Department of Computer Science and Technology, Shandong University, China
jguo@sdu.edu.cn

Daming Zhu
Department of Computer Science and Technology, Shandong University, China
dmzhu@sdu.edu.cn

Binhai Zhu
Gianforte School of Computing, Montana State University, Bozeman, MT 59717, USA
bhz@montana.edu

Abstract

The Maximal Strip Recovery problem (MSR) and its complementary (CMSR) are well-studied NP-hard problems in computational genomics. The input of these dual problems are two signed permutations. The goal is to delete some gene markers from both permutations, such that, in the remaining permutations, each gene marker has at least one common neighbor. Equivalently, the resulting permutations could be partitioned into common strips of length at least two. Then MSR is to maximize the number of remaining genes, while the objective of CMSR is to delete the minimum number of gene markers. In this paper, we present a new approximation algorithm for the Complementary Maximal Strip Recovery (CMSR) problem. Our approximation factor is 2, improving the currently best 7/3-approximation algorithm. Although the improvement on the factor is not huge, the analysis is greatly simplified by a compensating method, commonly referred to as the non-oblivious local search technique. In such a method a substitution may not always increase the value of the current solution (it sometimes may even decrease the solution value), though it always improves the value of another function seemingly unrelated to the objective function.

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

Keywords and phrases Maximal strip recovery, complementary maximal strip recovery, computational genomics, approximation algorithm, local search

Digital Object Identifier 10.4230/LIPIcs.CPM.2019.5

Funding This research is supported by National Natural Science Foundation of China, No. 61472222, 61732009, 61761136017, 61628207.

1 Introduction

Maximal Strip Recovery (MSR) is a problem originally proposed to eliminate noise and ambiguities in genomic maps [4, 15]. In comparative genomics, a genetic map (interchangeably, a signed permutation) is represented by a sequence of n distinct gene markers (interchangeably, letters). A gene marker can appear in two different genomic maps, either positively or negatively. A *strip* (or, syntenic block) is a sequence of distinct markers that appears as subsequences in two maps, either directly or in a reversed and negated form.

Given two genetic maps G_1 and G_2 of length n, the problem *Maximal Strip Recovery* (MSR) [4, 15] is to find two subsequences of d strips (each of length at least two), denoted as $G_i^\star$, for $i = 1, 2$, and find two signed permutations π_i of $\langle 1, \ldots, d\rangle$, such that each sequence

$G_i^\star = S_{\pi_i(1)} \dots S_{\pi_i(d)}$ (here S_{-j} denotes the reversed and negated sequence of S_j) is a subsequence of G_i, and the total length of these S_j's is maximized. Intuitively, those gene markers not included in $G_1^\star$ and $G_2^\star$ are noisy and ambiguous markers. The complementary problem of deleting the minimum number of noisy and ambiguous markers to have a feasible solution (i.e., every remaining marker must be in some strip) is called the *Complementary Maximal Strip Recovery*, which will be abbreviated as CMSR. We illustrate an example in Fig. 1. In this example, each integer in G_1 and G_2 represents a gene marker.

$$
\begin{aligned}
G_1 &= \langle 1,2,3,4,5,6,7,8,9,10,11,12,13\rangle \\
G_2 &= \langle -9,-4,-8,-7,-6,1,2,3,-13,-12,-10,-5,-11\rangle \\
S_1 &= \langle 1,2,3\rangle \\
S_2 &= \langle 6,7,8,9\rangle \\
S_3 &= \langle 11,12,13\rangle \\
\pi_1 &= \langle 1,2,3\rangle \\
\pi_2 &= \langle -2,1,-3\rangle \\
G_1^\star &= \langle 1,2,3,6,7,8,9,11,12,13\rangle \\
G_2^\star &= \langle -9,-8,-7,-6,1,2,3,-13,-12,-11\rangle
\end{aligned}
$$

Figure 1 An example for the problem MSR and CMSR. MSR has a solution size of ten (with $d = 3$ strips in $G_1^\star$ and $G_2^\star$; i.e., (1,2,3),(6,7,8,9) and (11,12,13)). CMSR has a solution size of three: the deleted markers are 4,5 and 10.

It was shown in [14] that both (the decision versions of) MSR and CMSR are NP-complete. (Readers are referred to [5] for the basic concepts in algorithms and NP-completeness.) A bit later, MSR was shown to be APX-hard [2, 9] and CMSR was also shown to be APX-hard [10]. For the positive results, in [4, 15], some heuristic approaches based on Max Independent Set and Max Clique were proposed and shown to be effective. In [3], a factor-4 polynomial-time approximation algorithm was proposed for MSR. In [7], a factor-3 polynomial-time approximation algorithm was proposed for CMSR and an $O^*(3^k)$ FPT algorithm, where k is the parameter representing the minimum number of deleted genes, was also presented for CMSR. Currently, the best approximation factor for CMSR is 2.33 [13] and the best FPT algorithmic bound is $O(2.36^k n^2)$ [1]. In 2014, Jiang and Zhu showed that CMSR admits a kernel of size $78k$ [8]. This kernel was improved to $58k$ more recently by Hu *et al.* [6] and then to $42k$ by Li *et al.* [12]. Combined with these kernel bounds, CMSR can be solved in $O(n^2 + 2.36^k k^2)$ time.

In this paper, we devise a new 2-approximation algorithm for the CMSR problem by a non-oblivious local search technique, initially proposed by Khanna *et al.* [11]. During the non-oblivious local search process, a substitution (or local update) may not always increase the value of the current solution, it sometimes makes the value unchanged or even decreased. (But it always improves the value of a function which is seemingly unrelated to the objective function – that is probably how the algorithm is named "non-oblivious".) In other words, some strips with larger priority are preferred even though they cannot lead to a local minimum solution. While the idea is simple, the analysis is very involved. We hope to see more applications of this technique for problems in computational biology.

2 Preliminaries

We first present some formal definitions. Let $G = \pi_1\pi_2\cdots\pi_n$ be a (signed) permutation and let $\bar{G}$ be the reversed version of G, with all elements negated. A *substring* of G is a consecutive segment in G, $\pi_i\pi_{i+1}\cdots\pi_j$, such that $1 \le i < j \le n$ and its length is denoted as $|\pi_i\pi_{i+1}\cdots\pi_j| = j - i + 1$. A *subsequence* of G is a sequence of letters $\pi_{l_1}\pi_{l_2}\cdots\pi_{l_k}$ such that $1 \le l_1 < l_2 \cdots < l_k \le n$.

Let G_1 and G_2 be two input (signed) permutations over the same set of alphabet (letters). (We assume that G_1 is the identity permutation I_n over $[1..n]$ throughout this paper.) A (common) *strip*, $\pi_{l_1}\pi_{l_2}\cdots\pi_{l_j}$ $(2 \le j \le n)$, is a common subsequence between G_1 and G_2 (or $\bar{G}_2$), with at least two letters, and its length is j. An *existing* common strip is a common substring between G_1 and G_2 (or $\bar{G}_2$), again with length at least two. Sometimes we also use $S = [a, b]$ to denote a common strip, with a and b being its ending letters, and $|S| = |[a, b]|$ to denote its length. An *adjacency* is a strip of length two. Hence, a pair of common strips of length l has $l - 1$ adjacencies. In Figure 1, the strip $(1, 2, 3)$ appears positively in G_2 and the strip $(6, 7, 8, 9)$ appears in reversed and negated form in G_2 (or, appears positively in $\bar{G}_2$); moreover, $(1, 2, 3) = [1, 3]$ is an existing common strip with two adjacencies.

The **Complementary Maximal Strip Recovery** (*CMSR*) problem is formally defined as follows: Given two signed permutations G_1 and G_2 over the same set of alphabet of size n, delete the minimum number of letters from G_1 and G_2 to obtain $G_1^\star$ and $G_2^\star$ such that the remaining letters are all in some common strips; moreover, each common strip is a substring in $G_1^\star$ and $G_2^\star$.

Two letters, a and b, form a *candidate adjacency* (a, b), if ab or $-b - a$ is a subsequence of G_1 and G_2. Let $IN_1(a, b)$ and $IN_2(a, b)$ be the set of (interior) letters that appears in between a and b in G_1 and G_2 respectively. If $|IN_1(a, b)|+|IN_2(a, b)| = i$, then (a, b) is called an *i-candidate* adjacency. Obviously, a 0-candidate adjacency is an adjacency from a pair of existing common strips of G_1 and G_2. An i-candidate adjacency, say (a, b), becomes an adjacency whenever the letters in $IN_1(a, b)$ and $IN_2(a, b)$ are deleted. In Figure 1, $(11, 12)$ is initially a 2-candidate adjacency and after the two markers 5 and 10 are deleted it becomes a strip (or a valid adjacency). The main idea of our algorithm is to identify some candidate adjacencies and delete the letters in between them.

3 Algorithm Description

In this section, we show the details of our algorithms. Since candidate adjacencies eventually form the final common strips after deleting the interior letters, the main idea of our algorithm is to identify some candidate adjacencies.

3.1 Preprocessing

Firstly, the algorithm preprocesses the two input permutations to identify some special subsequences, which could be put into the common strip components directly.

▶ **Definition 1** (*r*-candidate subsequence). A subsequence $a_1a_2\cdots a_{r-1}a_r$ of G_1 and G_2 (or $\bar{G}_2$), $r \ge 3$, is an *r-candidate subsequence,* if (a_i, a_{i+1}) is a 1-candidate adjacency, $1 \le i \le r-1$.

A letter c can *left-attach* (resp. *right-attach*) to a common strip $[a, b]$, if (c, a) (resp. (b, c)) is a candidate adjacency, then a new common strip $[c, b]$ (resp. $[a, c]$) is generated, while deleting the letters in $In_1(c, a) \cup In_2(c, a)$ (resp. $In_1(b, c) \cup In_2(b, c)$). Throughout Algorithm 1, let CS_0 be the current set of common strips. Initially, CS_0 is empty.

Algorithm 1 Preprocessing.

1: Put all the existing common strips into CS_0.
2: Compute a maximal set of non-overlapping r-candidate subsequences for $r \geq 3$ and put them in CS_0.
3: **for** each common strip $[a, b] \in CS_0$ **do**
4: **while** there exists a letter c, such that (c, a) or (b, c) is a 1-candidate adjacency **do**
5: Delete the letter in $In_1(c, a) \cup In_2(c, a)$ (or $In_1(b, c) \cup In_2(b, c)$) from G_1 and G_2, delete $[a, b]$ from CS_0, and add $[c, b]$ (or $[a, c]$) to CS_0.
6: **end while**
7: **end for**

Note that keeping existing common strips immediately implies that the factor 2 is the best we could have. For example, $G_1 = uabv \cdot xy$ and $G_2 = -v - u \cdot -yab - x$, keeping ab can only have an approximation factor 2 (as the optimal solution for CMSR is to delete a and b). The reason why we cannot keep 2-candidate subsequences is that they could force the approximation factor to be beyond 2. For instance, $G_1 = uv \cdot xa_1ya_2z$ and $G_2 = -va_1a_2 - u \cdot -z - y - x$, keeping a_1a_2 would give us an approximation factor of 2.5 (as we need to delete 5 letters, while the optimal solution is to delete a_1 and a_2). We now use CS_0 to denote the set of common strips in CS_0 found by the preprocessing Algorithm 1 and will not alter CS_0 henceforth. We then assign $CS \leftarrow CS_0$ and try to improve CS. Next, we show how to obtain more common strips.

3.2 How to generate a common strip

A letter is *matched* if it belongs to a common strip in CS, otherwise it is *unmatched*. A matched letter could have either one or two adjacent letters in a common strip. For a matched letter b, which has exactly one adjacent letter, say a, such that (a, b) is a p-candidate adjacency, then b is of *type-p*. For a matched letter b, which has exactly two adjacent letters, say a and c, such that (a, b) is a p-candidate adjacency and (b, c) is a q-candidate adjacency, then b is of *type*-$\min\{p, q\}$.

The letters in $In_1(a, b) \cup In_2(a, b)$ of a candidate adjacency (a, b) are *caught* in both G_1 and G_2, if a and b do form an adjacency; otherwise they are *released*. There are three ways to generate a common strip in our algorithm.

(I) ***Join***: generate a common strip $[a, b]$ from a p-candidate adjacency (a, b), where $p \leq 3$.

(II) ***Attach***: generate a common strip $[a, c]$ of length 3 by right-attaching a letter c to a common strip $[a, b]$ of length 2, such that (b, c) is a p-candidate adjacency, where $p \leq 2$. Moreover, if (a, b) is a 3-candidate adjacency, then delete a from $[a, c]$ to obtain the strip $[b, c]$, to release the 3 letters in $IN_1(a, b) \cup IN_2(a, b)$. The case that (c, a) is a p-candidate adjacency, where $p \leq 3$, is similar.

(III) ***Split***: generate two common strips $[a, b]$ and $[c, d]$, both of length 2, by right-attaching a letter d to a common strip $[a, b, c]$ of length 3, such that (c, d) is a p-candidate adjacency, where $p \leq 3$. The case that (d, a) is a p-candidate adjacency, where $p \leq 3$, is similar.

▶ **Property 1.** *Each common strip generated by Join, Attach, and Split operations is of length 2 or 3. Moreover, if* $[a, b]$ *is a common strip of length 2, then* (a, b) *is a* p*-candidate adjacency with* $p \leq 3$*; if* $[a, b, c]$ *is a common strip of length 3, then* (a, b) *is a* p*-candidate adjacency and* (b, c) *is* q*-candidate adjacency with* $p, q \leq 2$.

3.3 The Non-oblivious Local Search Algorithm

The main idea of our algorithm is a non-oblivious local search, which is outlined as Algorithm 2. The algorithm improves CS iteratively according to the objective function F. Let b_p be the number of type-p letters in the current solution, where $p \in \{1, 2, 3\}$, Define

$$F = 15b_1 + 5b_2 + b_3.$$

Generally, the algorithm adopts local substitutions, which means substituting exactly one current strip, say S, in CS with some other strips, provided that the value of F could be increased.

Let $CS = \{S_1, S_2, \cdots, S_m\}$ be the current set of common strips, and G'_1 and G'_2 be the current common strip components. For a common strip, say S_j, let S_i and S_k be its proceeding and following common strips in G'_1, and $S_{i'}$ and $S_{k'}$ be its proceeding and following common strips in G'_2 respectively. Define four letters to be $L^1_{S_j}$, $L^2_{S_j}$, $R^1_{S_j}$, $R^2_{S_j}$ as follows. If the number of letters appear in between S_i and S_j is less than 4 in G_1, then $L^1_{S_j}$ is the letter to the immediate right of S_i; otherwise, $L^1_{S_j}$ is the fourth letter to the left of S_j. The other three letters can be defined similarly.

CS could be improved by performing the following three operations iteratively.

1. *0-substitution*: Generate new common strips according to (I), (II) and (III), provided that the value of F can be increased.
2. *1-substitution*: Substitute a common strip S by some other strip, which can be searched from L^1_S to R^1_S in G_1 and from L^2_S to R^2_S in G_2 and according to (I), (II) and (III), provided that the value of F can be increased.
3. *2-substitution*: Substitute two consecutive common strips S_u and S_v in G'_1 (resp. G'_2) by two other strips, which can be searched from $L^1_{S_u}$ to $R^1_{S_v}$ in G_1 and from $L^2_{S_u}$ to $R^2_{S_u}$ as well as $L^2_{S_v}$ to $R^2_{S_v}$ in G_2 according to (I), (II) and (III), provided that the value of F can be increased.

The pseudo-code of our algorithm is shown in **Algorithm 2**.

Algorithm 2 CMSR by Non-oblivious-Local-Search.

```
Call Algorithm 1: Preprocessing.
while (0-substitution, 1-substitution, or 2-substitution can be applied) do
    Apply a 0-substitution, 1-substitution, or 2-substitution to improve CS.
end while
```

▶ **Theorem 2.** *The algorithm* CMSR by Non-oblivious-Local-Search *runs in* $O(n^2)$ *time.*

Proof. First of all, it is easy to see that Algorithm 1 runs in $O(n^2)$ time. It takes at most $O(n)$ time to perform a 0-substitution. A 1-substitution on a common strip S tries to identify all possible $1, 2, 3$-candidate adjacencies formed by letters from L^1_S to R^1_S in G_1 and from L^2_S to R^2_S in G_2. There are at most 21 distinct letters in this range if S is of length 2, and at most 23 distinct letters if S is of length 3. Hence the number of possible $1, 2, 3$-candidate adjacencies is bounded by a constant, and the number of its combination is also bounded by a constant. In this case, a candidate adjacency can be found in constant time. Similar argument holds for a 2-substitution.

The number of common strips in CS is obviously bounded by n. It remains to count the time spent on the While-Loop. Note that the While-Loop runs while CS is updated/refreshed, in other words, each round of the While-Loop implies a feasible substitution, and F must be

increased. Since the value of F increases by at least one each time, and the maximum value of F is bounded by $15n$, the While-Loop runs at most $15n$ times.

Consequently, it takes $O(n)$ time to find/perform a feasible substitution. And this procedure loops $O(n)$ times. Therefore the time complexity of **Algorithm 2** is $O(n^2)$. ◀

4 Performance Analysis

In this section, we show that **Algorithm 2** returns a 2-approximation for the complementary maximum strip recovery problem. To analyze the performance of **Algorithm 2**, we should compare it with the optimal solution. Here, we adopt the amortized analysis method.

Let $G_1 = [\pi_1\pi_2\cdots\pi_n] = I_n$ and $G_2 = [\tilde{\pi}_1\tilde{\pi}_2\cdots\tilde{\pi}_n]$ be two (signed) permutations over $[1..n]$. Let G_1^* and G_2^* be the optimal common strip components, each of which is composed of the set CS^* of common strips. Let OPT be the set of letters which are deleted from both G_1 and G_2 to obtain G_1^* and G_2^* respectively. As aforementioned, CS is the set of common strips computed by **Algorithm 2**, which constitute two feasible common strip components G_1' and G_2'. Let ALG be the set of letters which are deleted to obtain G_1' and G_2'. Without causing confusion, CS and CS^* can also be viewed as the sets of adjacencies, as well as the sets of preserved letters. Then we have, $|OPT| + |CS^*| = |ALG| + |CS| = n$.

Next, we review a property of the optimal solution presented in [7].

▶ **Lemma 3.** *There exists an optimal solution OPT, such that, for every existing common strip S of length 2 or greater, (1) S is either totally contained in OPT, or totally disjoint with OPT; (2) if S is of length 4 or greater, then it is totally disjoint with OPT.*

This lemma and its proof also appear in [7], so we omit the proof here. The idea, following the example right after Algorithm 1, is that while existing length-2 or length-3 common strips could be deleted in any optimal solution, when the existing common strips are of length at least 4 then there is some optimal solution which keeps them. For example, $G_1 = xabcdy \cdot uv$ and $G_2 = -y - x \cdot -vabcd - u$, then the optimal solution for MSR could either be $abcd$ (by keeping the existing length-4 common strip $abcd$, the corresponding CMSR solution is to delete u, v, x and y), or with two strips xy and uv (the corresponding CMSR solution is to delete a, b, c and d).

Now, consider the imaginary permutations whenever we reinsert the letters of $OPT \cap CS$ back into G_1^* and G_2^*. Some common strips could be broken into either shorter blocks or *isolates* (e.g., a single letter in either G_1 or G_2 which does not form adjacencies with its neighbors). For instance, the letters 9 and 11 in Fig. 1 are both isolates.

Without of loss of generality, we only focus on the case when the adjacency (a, b) (and (x, y)) appears positively in G_1 and G_2. The other case when it appears in reversed and negated form in G_2 is similar. But we omit the details.

▶ **Definition 4.** *Let (a, b) be an adjacency of a common strip in CS, where $a = \pi_{i_1} = \tilde{\pi}_{i_2}$ and $b = \pi_{j_1} = \tilde{\pi}_{j_2}$. Let (x, y) be an adjacency of a common strip in CS^*, where $x = \pi_{i_1^*} = \tilde{\pi}_{i_2^*}$ and $y = \pi_{j_1^*} = \tilde{\pi}_{j_2^*}$.*

*When $(a, b) = (x, y)$, we say that (a, b) **fully occupies** (x, y).*

*Let $\sqcap$ be the operation for computing the intersection of two intervals, which could be closed (i.e., in the form $[i, j]$) or open (i.e., in the form (i, j)). If $[i_1, j_1] \sqcap [i_1^*, j_1^*] = [i_1^*, i_1^*] = [j_1, j_1]$ or $[i_2, j_2] \sqcap [i_2^*, j_2^*] = [i_2^*, i_2^*] = [j_2, j_2]$, we say that (a, b) **half-occupies** (x, y) via the letter b, then y belongs to the **occupying set** of (a, b), denoted as $O[(a, b)]$. (The case (a, b) **half-occupies** (x, y) via a can be defined symmetrically, in which x belongs to $O[(a, b)]$.)*

Otherwise, if $(i_1, j_1) \sqcap (i_1^*, j_1^*) \neq \emptyset$ *or* $(i_2, j_2) \sqcap (i_2^*, j_2^*) \neq \emptyset$*, we say that* (a, b) ***breaks*** (x, y)*, then both* x *and* y *belong to the* ***breaking set*** *of* (a, b)*, denoted as* $B[(a, b)]$*.*

▶ **Lemma 5.** *Let* (x, y) *be an adjacency in* CS^**, but not in* CS*. Then* (x, y) *is either broken or half-occupied by some adjacencies in* CS*, or* (x, y) *is a p-candidate adjacency with* $p \geq 4$*.*

Proof. Since otherwise, **Algorithm 2** would keep running as the current solution can be improved. ◀

The ***occupying set*** of a common strip $S = [a, \ldots, b]$ in CS, denoted as $O[S]$, is the set of all letters from the adjacencies in CS^* which are half-occupied by $(a, \bullet)$ via a or by $(\bullet, b)$ via b, i.e., $O[S] = O[(a, \bullet)] \cup O[(\bullet, b)]$.

The ***breaking set*** of a common strip S in CS, denoted by $B[S]$, is the set of all letters from adjacencies in CS^*, which are broken by adjacencies of S, i.e., $B[S] = \cup_{(a,b)\subseteq S} B[(a, b)]$. It follows from the definition that, for each common strip S in CS, $O[S] \cap B[S] = \emptyset$.

The ***auxiliary set*** of a common strip S in CS, denoted by $U[S]$, is the set of all letters such that (1) not in CS, (2) appearing in between adjacencies of CS^*, which are broken or half-occupied by adjacencies of S.

Let $CS^*_{\geq 4}$ be the set of letters from p-adjacencies of CS^* with $p \geq 4$, which are not broken or half-occupied by any common strip of CS. Let $U^*_{\geq 4}$ be the set of letters which appear in between adjacencies formed by letters in $CS^*_{\geq 4}$. Viewing CS and CS^* as set of letters, we have $ALG \cup CS = OPT \cup CS^*$. Consequently

$$ALG - OPT = CS^* - CS = \cup_{S\in CS}(B[S] \cup O[S]) \bigcup CS^*_{\geq 4}.$$

From the definition, we know that the union of the auxiliary sets are letters in $OPT \cap ALG$ which appear in between adjacencies of $CS^* - CS^*_{\geq 4}$, i.e.,

$$OPT \cap ALG \supseteq (\cup_{S\in CS} U[S]) \bigcup U^*_{\geq 4}.$$

We also have,

$$OPT - ALG = CS - CS^* = \cup_{S\in CS}(S - CS^*).$$

The approximation factor can be described as

$$\frac{|ALG|}{|OPT|} = \frac{|ALG - OPT| + |OPT \cap ALG|}{|OPT - ALG| + |OPT \cap ALG|} \leq 2$$

which is equivalent to,

$$\begin{aligned}
\frac{|ALG|}{|OPT|} &\leq \frac{|ALG - OPT| - |OPT \cap ALG|}{|OPT - ALG|} \\
&\leq \frac{|\cup_{S\in CS}(B[S] \cup O[S]) \bigcup CS^*_{\geq 4}| - |(\cup_{S\in CS} U[S]) \bigcup U^*_{\geq 4}|}{|\cup_{S\in CS}(S - CS^*)|} \\
&= \frac{|\cup_{S\in CS}(B[S] \cup O[S])| + |\bigcup CS^*_{\geq 4}| - |(\cup_{S\in CS} U[S]) \bigcup U^*_{\geq 4}|}{|\cup_{S\in CS}(S - CS^*)|} \leq 2 \qquad (1)
\end{aligned}$$

Since each letter in $OPT \cap ALG$ appears in G_1 and G_2 exactly once respectively, we have,

$$|(\cup_{S\in CS} U[S]) \bigcup U^*_{\geq 4}| = \frac{\sum_{S\in CS} |U[S]| + |U^*_{\geq 4}|}{2}.$$

Then, it is sufficient to show that

$$(1) = \frac{|\cup_{S\in CS}(B[S] \cup O[S])| - \frac{\sum_{S\in CS} |U[S]|}{2} + |\bigcup CS^*_{\geq 4}| - \frac{|U^*_{\geq 4}|}{2}}{|\cup_{S\in CS}(S - CS^*)|} \leq 2 \qquad (2)$$

Note that a letter may belong to more than one breaking and half-occupying sets. If we assign a weight $\omega(-)$ to each letter of each set, then (2) can be rewritten as

$$(2) \quad = \quad \frac{\sum_{S\in CS}(\omega(B[S]) + \omega(O[S]) - \frac{|U[S]|}{2}) + |\bigcup CS^*_{\geq 4}| - \frac{|U^*_{\geq 4}|}{2}}{\sum_{S\in CS}|S - CS^*|} \leq 2 \tag{3}$$

▶ **Theorem 6.** *The approximation factor will not be greater than two, if the following two conditions hold:*

1. $|\bigcup CS^*_{\geq 4}| - |U^*_{\geq 4}|/2 \leq 0$, *and*
2. *for each* $S \in CS$, $(\omega(B[S]) + \omega(O[S]) - |U[S]|/2)/|S - CS^*| \leq 2$.

The former condition holds from the following lemma straightforwardly.

▶ **Lemma 7.** *Let* (x, y) *be a* p*-adjacency of* CS^* *with* $p \geq 4$*, which is not broken or half-occupied by any adjacency of* CS*, being not able to return* (x, y) *as an adjacency will not result in an approximation factor greater than 2.*

Proof. Since the optimal solution includes the p letters appearing in between x and y, besides these p letters, the approximated solution also includes x and y. Then we have $2 - p/2 \leq 0$, in light of $p \geq 4$. ◀

It remains to assign weights to the letter of the breaking and half-occupying sets. Note that a letter cannot appear in two half-occupying sets.

Weight Assignment.

- **(I)** For each $S \in CS_0$, each letter of $B[S]$ bears a weight of 1; each letter of $O[S]$ bears a weight of 1 if it does not appear in any other set, and bears a weight of 0 if it also appears in some other breaking sets.
- **(II)** If a letter appears in exactly one breaking set, then this appearance bears a weight of 1; if it appears in two or more breaking sets, each of its appearance bears a weight of $\frac{1}{2}$.
- **(III)** If a letter appears in exactly one half-occupying set, then this appearance bears a weight of 1.
- **(IV)** If a letter appears in the half-occupying set $O[S](S \notin CS_0)$, as well as in the breaking set $B[S']$, then $O[S]$ bears a weight of $-\frac{1}{2}$, and $B[S']$ bears a weight of $3/2$.
- **(V)** If a letter appears in the half-occupying set $O[S](S \notin CS_0)$, as well as two breaking sets $B[S']$ and $B[S'']$, then $O[S]$ bears a weight of $-\frac{1}{2}$, and $B[S']$ bears a weight of 1 and $B[S'']$ bears a weight of $\frac{1}{2}$.

▶ **Lemma 8.** $\sum_{S\in CS}(\omega(B[S]) + \omega(O[S])) \geq |\cup_{S\in CS}(B[S] \cup O[S])|$.

Proof. It can be verified that, under the above weight assignment, each letter of $\cup_{S\in CS}(B[S] \cup O[S])$ has a total weight of at least 1. ◀

Next, we show that condition (2) is satisfied. We say that a common strip $S \in CS$ is *safe*, if $\frac{|B[S]|}{|S-CS^*|} \leq 2$. An algorithm is *safe* if all the common strips generated at the end of the algorithm are safe.

4.1 Algorithm 1 is Safe

We first try to show that all the common strips founded by Algorithm 1 are safe.

▶ **Lemma 9.** *The existing common strips are safe.*

Proof. Assume that S is an existing common strip of G_1 and G_2. If $S \subseteq CS^*$, we are done. If not, according to Lemma 3, it satisfies that $|S - CS^*| = |S| \geq 2$. In this case, reinserting S back into G_1^* and G_2^*, would break at most 2 adjacencies in CS^*, thus $|B(S)| \leq 4$. then we have $|B[S]|/|S - CS^*| \leq 4/2 = 2$. ◀

The following example shows that (keeping) the existing common strips would be safe even for the worst case. Let $G_1 = 1ab2 \cdot 34$ and $G_2 = -2 - 1 \cdot -4ab - 3$. Keeping $S = ab$ would imply deleting the four letters $\{1, 2, 3, 4\}$, while the optimal solution is to delete $\{a, b\}$. Note that in this example, $B[S] = \{1, 2, 3, 4\}$, $S - CS^* = \{a, b\}$, hence $|B[S]| = 4$ and $|S - CS^*| = 2$; moreover,

$$|B[S]|/|S - CS^*| = 2.$$

▶ **Lemma 10.** *Every 3-candidate subsequence is safe.*

Proof. Let $T = abc$ be a 3-candidate subsequence. From **Definition 1**, both (a, b) and (b, c) are 1-candidate adjacencies. Assume that the letter x appears in between a and b in G_1 or G_2, and the letter y appears in between b and c in G_1 or G_2. There are 8 cases according to whether or not a, b, c belong to OPT.

1. $a, b, c \notin OPT$. $[a, b, c]$ becomes a common strip after deleting x and y from both G_1 and G_2, according to Lemma 9, $[a, b, c]$ is safe.
2. $\{a, b, c\} \subseteq OPT$. Reinserting $\{a, b, c]\}$ back into G_1^* and G_2^* would break at most 4 adjacencies, thus $|B([a, b, c])| \leq 6$, and $|B([a, b, c])|/|(\{a, b, c\} - CS^*)| \leq 6/3{=}2$.
3. $\{a, b\} \subseteq OPT$. Reinserting $[a, b]$ back into G_1^* and G_2^* would break at most 2 adjacencies, thus $|B([a, b, c])| \leq 4$, and $|B([a, b, c])|/|(\{a, b, c\} - CS^*)| \leq 4/2{=}2$.
4. The case when $\{b, c\} \subseteq OPT$ is symmetric to the case $\{a, b\} \subseteq OPT$.
5. $a \in OPT$, then $y \in OPT$. Reinserting a back into G_1^* and G_2^* would break at most one adjacency, thus $|B([a, b, c])| \leq 2$, and $|B([a, b, c])|/|(\{a, b, c\} - CS^*)| \leq 2/1{=}2$.
6. The case when $c \in OPT$ is symmetric to the case $a \in OPT$.
7. $b \in OPT$. Reinserting b back into G_1^* and G_2^* would break at most one adjacency (x, y), thus $|B([a, b, c])| \leq 2$, and $|B([a, b, c])|/|(\{a, b, c\} - CS^*)| \leq 2/1{=}2$.
8. $\{a, c\} \subseteq OPT$ but not b, then there exists a p-adjacency $(b, d) \in CS^*$. Since $In_1(b, d) \cup In_2(b, d) \supseteq In_1(b, c) \cup In_2(b, c)| = \{y\}$, we could obtain another optimal solution by replacing c by d in CS^*. ◀

▶ **Lemma 11.** *Generate a common strip $S' = [a, \ldots, b, c]$ by attaching a letter c to a safe common strip $S = [a, \ldots, b]$ via a 1-candidate adjacency (b, c), then S' is safe.*

Proof. Since $S = [a, \ldots, b]$ is safe, we have $\frac{|B(S)|}{|S - CS^*|} \leq 2$. Assume that the letter x appears in between b and c in G_1 or G_2. There are 2 cases according to whether or not $c \in OPT$:

(1) $c \in OPT$. Reinserting c back into G_1^* and G_2^* would break at most one adjacency (x, y), which is not broken by a, b. Thus, $|B(S') - B(S)| \leq 2$. Note that $|S' - CS^*| = |S - CS^*| + 1$, then $\frac{|B(S')|}{|S' - CS^*|} \leq \frac{|B(S)|+2}{|S - CS^*|+1} \leq 2$.

(2) $c \notin OPT$. In this case, either $x \in OPT$ or $x \in B(S)$. In either case, $B(S') = B(S)$ and $S - CS^* = S' - CS^*$. ◀

From Lemma 9,10,11, we know that all the common strips of CS_0 are safe.

▶ **Lemma 12.** *For each* $S \in CS_0$, $\frac{\omega(B[S])+\omega(O[S])-|U[S]|/2}{|S-CS^*|} \leq 2$.

Proof. It suffices to show that $\omega(O[S]) - |U[S]|/2 \leq 0$. Assume that $S = [a, \cdots, b]$, (b, c) is an adjacency in CS^*, which means $c \in O[S]$. By the **Weight Assignment** scheme, if (b, c) is broken by some other adjacencies in CS, then c bears a weight of 0 in $O[S]$. If (b, c) is not broken by any other adjacency of CS, then c bears a weight of 1 in $O[S]$. According **Algorithm 1**, (b, c) is a p-candidate adjacency with $p \geq 2$, and these p letters are all in $U[S]$. Thus, $1 - p/2 \leq 0$. ◀

4.2 Algorithm 2 is Safe

Aside from the common strips in CS_0, we now focus on the other common strips in CS which are of length either 2 or 3. If it is of length 2 and it is also an i-adjacency, we say that it is an *i-common strip*; if its length is 3 and it has two consecutive adjacencies: an i-adjacency and a j-adjacency, we say that it is an *$i \bowtie j$-common strip*. From Property 1, there are five types of common strips: 1-common strip, 2-common strip, 3-common strip, $1 \bowtie 2$-common strip, $2 \bowtie 2$-common strip.

Before showing that the common strips found by **Algorithm 2** fulfills condition 2, we first show some properties. The key idea is that, when **Algorithm 2** terminates, the value of F will not be increased by applying more 0,1,2-substitutions.

▶ **Lemma 13.** *At the termination of **Algorithm 2**, a 1-candidate adjacency will either become an adjacency in CS or be broken by some 1-adjacency in CS.*

Proof. Firstly, a 1-adjacency cannot be half-occupied by a 1-adjacency in CS. The reason is that, in this case, the two 1-adjacencies form a 3-candidate subsequence, which would have been handled by **Algorithm 1**.

Then we show that a 1-adjacency cannot only be half-occupied by adjacencies in CS. Assume that $S = [a, \cdots, b] \in CS$ and (b, c) is a 1-adjacency. Obviously, $S \notin CS_0$, thus $|S| = 2$ or $|S| = 3$, and $(\bullet, b)$ is a p-adjacency with $p \geq 2$. If $|S|$=2, then left-attaching c to S will increase the value of F definitely, if $|S| = 3$, then substituting S with $[a, \cdots]$ and $[b, c]$ will also increase the value of F. In case that $S' = [c, \cdots, d] \in CS$, then $|S'| = 2$ or $|S'| = 3$, and $(c, \bullet)$ is a q-adjacency with $q \geq 2$. keeping the common strip $[b, c]$ will increase the value of F.

If a 1-adjacency (b, c) is half-occupied by some adjacency A in CS and is also broken by another t-adjacencies A' of CS, then for $t \geq 2$, keeping the common strips $[b, c]$ will increase the value of F by at least $15 - 2 \times 5 = 5$. ◀

From Lemma 13 and the **Weight Assignment** scheme, if an adjacency is multiply broken, then each common strip breaking it bears a weight of 1 from it. If an adjacency is singly broken by $S \in CS$ and is also half-occupied by $S' \in CS$, then the common strip breaking it bears a weight of 3/2 from it, and the common strip half-occupying it bears a weight of $-1/2$ from it; moreover, this adjacency must be a p-adjacency with $p \geq 2$, which means that there could be a letter in $U[S]$.

▶ **Lemma 14.** *At the termination of **Algorithm 2**, a 2-adjacency will (1) either become an adjacency of CS, (2) or be broken by some adjacencies of CS, (3) or be half-occupied by two adjacencies of CS, (4) or be half-occupied by only one 1-adjacency of CS, which is in a common strip of CS_0 or a $1 \bowtie 2$-common strip.*

Proof. From Lemma 5, a 2-adjacency will either become an adjacency of CS or be broken or be half-occupied. Note that an adjacency can be half-occupied at most twice, then it is

sufficient to show that (4) holds. The reason is that if a 2-adjacency is only half-occupied by a 1-common strip, 2-common strip, 3-common strip, or a 2-adjacency in either a $1 \bowtie 2$-common strip or a $2 \bowtie 2$-common strip, then value of F will be increased by keeping it. ◀

▶ **Lemma 15.** *At the termination of **Algorithm 2**, if a p-adjacency in CS^* is broken by a q-adjacency in CS, where $q \leq 3$, then either $p \geq q$ or it is also broken or half-occupied by some other adjacencies in CS.*

Proof. Since otherwise, **Algorithm 2** will keep running as the current solution can be improved. ◀

▶ **Lemma 16.** *A p-common strip $S = [a, b]$, where $p = 1, 2, 3$, guarantees that* $\frac{\omega(B[S])+\omega(O[S])-|U[S]|/2}{|S-CS^*|} \leq 2$.

Proof. There are three cases: $|S - CS^*| = 2$, $|S - CS^*| = 1$, $|S - CS^*| = 0$.

(1) $|S - CS^*| = 2$. $|B[S]| \leq p + 4$ and $|O[S]| = 0$.

- **(1.1)** $p = 1$, there are $p + 2 = 3$ adjacencies broken by S. Since S is a common strip at the termination of **Algorithm 2** (i.e., no more local improvement is possible), at most one of these three adjacencies is a 1-candidate adjacency, which is singly broken by S. The other adjacencies must either be broken or half-occupied by some other adjacency in CS, or be p-candidate adjacencies with $p \geq 4$. For each of them, if it is half-occupied, $|B[S]| + |O[S]| \leq 4$. According to the **Weight Assignment** scheme, $B[S]$ bears a weight of $3/2$, which means some other adjacency in CS bears a weight of $-1/2$. Moreover, if it is broken, $B[S]$ bears a weight of 1; if it is a p-candidate adjacency with $p \geq 4$, then three letters are added to $U[S]$. In the worst case, we have $\frac{\omega(B[S])+\omega(O[S])-|U[S]|/2}{2} \leq \frac{p+4-p}{2} = 2$.
- **(1.2)** $p = 2$, there are at least $p + 1 = 3$ candidate adjacencies broken by S. As no local improvement is available, at most one of them could be a 2-candidate adjacency, which is singly broken by S. The others must either be broken or half-occupied by other adjacencies in CS, or be p-candidate adjacencies with $p \geq 4$. By an argument similar to (1.1), in the worst case, we have $\frac{\omega(B[S])+\omega(O[S])-|U[S]|/2}{2} \leq \frac{p+4-p}{2} = 2$.
- **(1.3)** $p = 3$, there are at least $p + 2 = 5$ candidate adjacencies broken by S. Similar to the above argument, as no local improvement is available, at least $p + 1$ candidate adjacencies must either be broken or half-occupied by other adjacencies in CS, or be p-candidate adjacencies with $p \geq 4$. Similar to the previous arguments, we have $\frac{\omega(B[S])+\omega(O[S])-|U[S]|/2}{2} \leq \frac{p+4-(p+1)}{2} \leq 2$.

(2) $|S - CS^*| = 1$. $|B[S]| \leq p + 2$ and $|O[S]| = 1$. In this case, according to the **Weight Assignment** scheme, $O[S]$ bears a weight of $-1/2$.

- **(2.1)** $p = 1$, then we are done, since $|B[S]| \leq 2$.
- **(2.2)** $p = 2$, there are two candidate adjacencies broken by S. As local improvement is not possible, at most one of them could be a 2-candidate adjacency. The 2-candidate adjacency bears a weight 2, while the other one bears a weight 1; moreover, the 2-candidate adjacency also implies that a letter is in $U[S]$. So we have, $\frac{\omega(B[S])+\omega(O[S])-|U[S]|/2}{2} \leq \frac{p+2-1/2-1-1/2}{1} = 2$.
- **(2.3)** $p = 3$, there are three candidate adjacencies broken by S. As local improvement cannot be performed further, at most one of them could be a 3-candidate adjacency. The 3-candidate adjacency bears a weight 2, while the other one bears a weight 1; moreover, the 3-candidate adjacency also implies that two letters are in $U[S]$. So we have, $\frac{\omega(B[S])+\omega(O[S])-|U[S]|/2}{2} \leq \frac{p+2-1/2-1-1-1/2}{1} = 2$.

(3) $|S - CS^*| = 0$. $|B[S]| \leq p$ and $|O[S]| = 2$. In this case, according to the **Weight Assignment** procedure, $O[S]$ bears a weight of -1.

(3.1) $p = 1$, then we are done, since $\omega(B[S]) + \omega(O[S]) \leq 0$.

(3.2) $p = 2$, there is one candidate adjacency broken by S, which could be multiply broken or be a p-candidate adjacency with $p \geq 2$. The former means that $B[S]$ bears a weight of 1, the latter means that there are two letters in $U[S]$. So we have, $\omega(B[S]) + \omega(O[S]) - |U[S]|/2 \leq 0$.

(3.3) $p = 3$, if there is one candidate adjacency broken by S, then it becomes the case (3.2). If there are two candidate adjacencies composed of three letters broken by S, then they could be multiply broken or be a p-candidate adjacency with $p \geq 3$. For each of them, if it is also broken by some common strips of CS_0, then $B[S]$ bears a weight of 0, and we are done; if it is a p-candidate adjacency with $p \geq 3$, then there are three letters in $U[S]$. If both of them are also broken by common strips not in CS_0, both adjacencies are not 1-candidate adjacencies (since otherwise, they become a 3-candidate subsequence). Thus, a new letter appears in $U[S]$. So we have, $\omega(B[S]) + \omega(O[S]) - |U[S]|/2 \leq 0$. ◀

▶ **Lemma 17.** *A $p \bowtie q$-common strip $S = [a, b, c]$, where $p = 1, 2$ and $q = 2$, guarantees that* $\frac{\omega(B[S])+\omega(O[S])-|U[S]|/2}{|S-CS^*|} \leq 2$.

Proof. Similar to Lemma 16, hence the details are omitted. ◀

We summarize the main result of this paper as follows.

▶ **Theorem 18.** *The algorithm* CMSR by Nonoblivious-Local-Search *approximates CMSR with a factor of 2, and it runs in $O(n^2)$ time.*

5 Concluding Remarks

We show a non-trivial application of non-oblivious local search for the CMSR problem. The local update step does not always increase the objective function. The difficulty has been assigning different weights for some potential common strips. We hope that this technique might be useful to other optimization problems in computational biology.

References

1 Laurent Bulteau, Guillaume Fertin, Minghui Jiang, and Irena Rusu. Tractability and approximability of maximal strip recovery. *Theoretical Computer Science*, 440-441:14–28, 2012. doi:10.1016/j.tcs.2012.04.034.

2 Laurent Bulteau, Guillaume Fertin, and Irena Rusu. Maximal Strip Recovery Problem with Gaps: Hardness and Approximation Algorithms. In *Algorithms and Computation, 20th International Symposium, ISAAC 2009, Honolulu, Hawaii, USA, December 16-18, 2009. Proceedings*, pages 710–719, 2009. doi:10.1007/978-3-642-10631-6_72.

3 Zhixiang Chen, Bin Fu, Minghui Jiang, and Binhai Zhu. On recovering syntenic blocks from comparative maps. *Journal of Combinatorial Optimization*, 18(3):307–318, 2009. doi:10.1007/s10878-009-9233-x.

4 Vicky Choi, Chunfang Zheng, Qian Zhu, and David Sankoff. Algorithms for the Extraction of Synteny Blocks from Comparative Maps. In *Algorithms in Bioinformatics, 7th International Workshop, WABI 2007, Philadelphia, PA, USA, September 8-9, 2007, Proceedings*, pages 277–288, 2007. doi:10.1007/978-3-540-74126-8_26.

5 M. R. Garey and David S. Johnson. *Computers and Intractability: A Guide to the Theory of NP-Completeness*. W. H. Freeman, 1979.

6 Shuai Hu, Wenjun Li, and Jianxin Wang. An Improved Kernel for the Complementary Maximal Strip Recovery Problem. In *Computing and Combinatorics - 21st International Conference, COCOON 2015, Beijing, China, August 4-6, 2015, Proceedings*, pages 601–608, 2015. doi:10.1007/978-3-319-21398-9_47.

7 Haitao Jiang, Zhong Li, Guohui Lin, Lusheng Wang, and Binhai Zhu. Exact and approximation algorithms for the complementary maximal strip recovery problem. *Journal of Combinatorial Optimization*, 23(4):493–506, 2012. doi:10.1007/s10878-010-9366-y.

8 Haitao Jiang and Binhai Zhu. A linear kernel for the complementary maximal strip recovery problem. *Journal of Computer and System Sciences*, 80(7):1350–1358, 2014. doi:10.1016/j.jcss.2014.03.005.

9 Minghui Jiang. Inapproximability of Maximal Strip Recovery. In *Algorithms and Computation, 20th International Symposium, ISAAC 2009, Honolulu, Hawaii, USA, December 16-18, 2009. Proceedings*, pages 616–625, 2009. doi:10.1007/978-3-642-10631-6_63.

10 Minghui Jiang. Inapproximability of Maximal Strip Recovery: II. In *Frontiers in Algorithmics, 4th International Workshop, FAW 2010, Wuhan, China, August 11-13, 2010. Proceedings*, pages 53–64, 2010. doi:10.1007/978-3-642-14553-7_8.

11 Sanjeev Khanna, Rajeev Motwani, Madhu Sudan, and Umesh V. Vazirani. On Syntactic versus Computational Views of Approximability. *SIAM Journal on Computing*, 28(1):164–191, 1998. doi:10.1137/S0097539795286612.

12 Wenjun Li, Haiyan Liu, Jianxin Wang, Lingyun Xiang, and Yongjie Yang. A 42k Kernel for the Complementary Maximal Strip Recovery Problem. In *Frontiers in Algorithmics - 11th International Workshop, FAW 2017, Chengdu, China, June 23-25, 2017, Proceedings*, pages 175–186, 2017. doi:10.1007/978-3-319-59605-1_16.

13 Guohui Lin, Randy Goebel, Zhong Li, and Lusheng Wang. An improved approximation algorithm for the complementary maximal strip recovery problem. *Journal of Computer and System Sciences*, 78(3):720–730, 2012. doi:10.1016/j.jcss.2011.10.014.

14 Lusheng Wang and Binhai Zhu. On the Tractability of Maximal Strip Recovery. *Journal of Computational Biology*, 17(7):907–914, 2010. (Correction: 18(1):129, Jan, 2011). doi:10.1089/cmb.2009.0084.

15 Chunfang Zheng, Qian Zhu, and David Sankoff. Removing Noise and Ambiguities from Comparative Maps in Rearrangement Analysis. *IEEE/ACM Transactions on Computational Biology and Bioinformatics*, 4(4):515–522, 2007. doi:10.1145/1322075.1322077.