

Exact and approximation algorithms for the complementary maximal strip recovery problem

Haitao Jiang* Zhong Li† Guohui Lin†‡ Lusheng Wang§ Binhai Zhu¶

November 12, 2010

Abstract

Given two genomic maps G_1 and G_2 each represented as a sequence of n gene markers, the *maximal strip recovery* (MSR) problem is to retain the maximum number of markers in both G_1 and G_2 such that the resultant subsequences, denoted as G_1^* and G_2^* , can be partitioned into the same set of maximal substrings of length greater than or equal to two. Such substrings can occur in the reversal and negated form. The *complementary* maximal strip recovery (CMSR) problem is to delete the minimum number of markers from both G_1 and G_2 for the same purpose, with its optimization goal exactly complementary to maximizing the total number of gene markers retained in the final maximal substrings. Both MSR and CMSR have been shown NP-hard and APX-hard. A 4-approximation algorithm is known for the MSR problem, but no constant ratio approximation algorithm for CMSR. In this paper, we present an $O(3^k n^2)$ -time fixed-parameter tractable (FPT) algorithm, where k is the size of the optimal solution, and a 3-approximation algorithm for the CMSR problem.

Keywords: Fixed-parameter tractable, approximation algorithm, amortized analysis

1 Introduction

In comparative genomics, one of the first steps is to decompose two given genomes into synthetic blocks — segments of chromosomes that are deemed homologous in the two input genomes. Many decomposition methods have been proposed, but they are very vulnerable to ambiguities and errors. A few years back, the *maximal strip recovery* (MSR) problem was formulated for eliminating noise and ambiguities in genomic maps, which are isolated points that do not co-exist with other points [4, 9]. In the more precise formulation, we are given two genomic maps G_1 and G_2 each represented as

*School of Computer Science and Technology, Shandong University. Jinan, Shandong, China. Email: htjiang@cs.montana.edu

†Department of Computing Science, University of Alberta. Edmonton, Alberta T6G 2E8, Canada. Email: zhong4@ualberta.ca

‡Email: guohui@ualberta.ca

§Department of Computer Science, City University of Hong Kong. Kowloon, Hong Kong, China. Email: cswangl@cityu.edu.hk

¶Department of Computer Science, Montana State University. Bozeman, Montana 59717-3880, USA. Email: bhz@cs.montana.edu

a sequence of n distinct gene markers, and we want to retain the maximum number of markers in both G_1 and G_2 such that the resultant subsequences, denoted as G_1^* and G_2^* , can be partitioned into the same set of maximal substrings of length greater than or equal to two. Each retained marker thus belongs to exactly one of these substrings, which can appear in the reversed and negated form and are taken as nontrivial chromosomal segments. The deleted markers are regarded as noise or errors.

The MSR problem, and its several close variants, have been shown NP-hard [8, 2, 3]. More recently, it is shown to be APX-complete [2, 6], admitting a 4-approximation algorithm [3]. This approximation algorithm is a modification of an earlier heuristics for computing a maximum clique (and its complement, a maximum independent set) [4, 9], to convert the MSR problem to computing the maximum independent set in t -interval graphs, which admits a $2t$ -approximation [1, 3]. In this paper, we investigate the complementary optimization goal to minimize the number of deleted markers — the *complementary MSR* problem, or CMSR for short. CMSR is certainly NP-hard, and was proven to be APX-hard recently [7]. Nevertheless, there is no known constant ratio approximation algorithm. We present here an $O(3^k n^2)$ -time exact bounded search tree algorithm for the problem, where k is the size of the optimal solution, and a 3-approximation algorithm. We want to point out that, a short fixed-parameter tractable (FPT) algorithm was proposed in [8] based on an incorrect lemma, which is corrected in this paper as Lemma 1. This lemma is also useful in our FPT algorithm and the approximation algorithm.

In the sequel, we use a lower case letter to denote a gene marker. A negation sign together with the succeeding gene indicate that the gene is in its reversal and negated form. We reserve the dot symbol for connection use, for example, $a \cdot b$ means gene b comes directly after gene a . When a common substring (also called *strip*, or *synthetic block*) of G_1 and G_2 is identified, we will (often) label it using a capital letter. We abuse this capital letter a bit to also denote the set of genes in the substring.

The rest of the paper is organized as follows. In Section 2, we present some structural properties of the optimal solutions to the CMSR problem, which will be used in the design of the FPT algorithm, presented in Section 3, and the design and the worst-case performance analysis of the 3-approximation algorithm, presented in Section 4. We conclude the paper with a few open questions in Section 5.

2 Structural properties of the CMSR problem

We first look at an example instance of the CMSR problem (also an instance of the MSR problem), in which $G_1 = \langle a, b, c, d, e, f, g, h, i, j, k, \ell \rangle$ and $G_2 = \langle -i, -d, -g, -f, h, a, c, b, -\ell, -k, -j, -e \rangle$ (we use commas to separate the gene markers for easier reading). By deleting markers $c, d, e,$ and h from both G_1 and G_2 , the resultant subsequences are $G_1^* = \langle a, b, f, g, i, j, k, \ell \rangle$ and $G_2^* = \langle -i, -g, -f, a, b, -\ell, -k, -j \rangle$. These two resultant subsequences can be decomposed into three maximal substrings $S_1 = a \cdot b$, $S_2 = f \cdot g \cdot i$ (appearing in the reversal and negated form in G_2^*), and $S_3 = j \cdot k \cdot \ell$ (appearing in the reversal and negated form in G_2^*). For this small instance, one can prove that the optimal solution to the MSR problem has size 8, and (consequently) the optimal solution to the CMSR problem has size 4.

Given any instance, in at most quadratic time, we can determine all maximal common substrings

in G_1 and G_2 . Note that the quadratic time could be improved to a linear time, with proper data structure such as suffix-tree. Note also that a substring and its reversed negated form are considered identical. Every letter in G_1 occurs in exactly one of these substrings. Some of these substrings have length greater than or equal to two, called *type-0* substrings; the others have length one, called *isolates*.

An optimal solution OPT to the instance (of the CMSR problem) is a minimum-size subset of letters that, deleting them from G_1 and G_2 gives the remainder subsequences denoted G_1^* and G_2^* , respectively, which can be partitioned into maximal substrings of length at least 2.

Lemma 1 *There exists an optimal solution OPT to the instance, such that*

- 1) *for each type-0 substring S , either $S \subset OPT$ or $S \cap OPT = \emptyset$;*
- 2) *if $|S| \geq 4$, then $S \cap OPT = \emptyset$.*

PROOF. Let OPT be an optimal solution. For a type-0 substring S , assume to the contrary that some but not all of its letters are in OPT . We know that the letters of $S - OPT$ appear consecutively in both G_1^* and G_2^* , and they form or participate in a single maximal substring, denoted as T . We may put letters of $S \cap OPT$ back to G_1^* and G_2^* according to their positions in G_1 and G_2 , respectively. These letters do not break but participate in the maximal substring T . This contradicts the optimality of OPT . Therefore, either $S \subset OPT$, or $S \cap OPT = \emptyset$.

If S has length of 4 or greater and $S \subset OPT$, we again put the letters of S back to G_1^* and G_2^* according to their positions in G_1 and G_2 , respectively. This added S , as a consecutive segment, might break into maximal substrings of G_1^* and G_2^* to give rise to at most 4 distinct letters that no longer belong to any maximal substrings of length at least 2. Since S becomes a (or part of a) maximal common substring, we can delete the (at least 4) letters of S from OPT while adding to OPT the (at most 4) letters that fall out of maximal substrings of length at least 2. The added letters certainly do not belong to any type-0 substrings. Therefore, this letter-swapping process gives another optimal solution that contains one less type-0 substring of length at least 4. Repeating the same argument if necessary, at the end we will achieve an optimal solution that do not contain any type-0 substring of length at least 4. \square

Lemma 1 fixes an erroneous claim made in [8], that OPT contains no type-0 substrings.

3 An exact bounded search tree algorithm

In this section, we consider solving CMSR with an FPT algorithm [5]. Basically, an FPT algorithm for a decision problem Π on whether or not there exists a solution of value (in our case, at most) k is an algorithm that solves the problem in $O(f(k)n^c) = O^*(f(k))$ time, where $f(\cdot)$ is any function only on k , n is the input size, and c is some fixed constant not related to k .

Given any instance, in the FPT algorithm to be described, it first determines all type-0 substrings. By Lemma 1, it will never delete any letter of those type-0 substrings of length at least 4 (*i.e.*, they are all retained in G_1^* and G_2^*). Some of these substrings, however, might get extended

by appending other letters, or merge into longer final maximal common substrings of G_1^* and G_2^* . Also by Lemma 1, in the sequel the algorithm considers maximal substrings of G_1 and G_2 as single units. For ease of presentation, all the type-0 substrings of length at least 4 are *marked*. The unmarked units are thus length-3 and length-2 type-0 substrings, and isolates. The FPT algorithm considers an isolate; it either deletes it from G_1 and G_2 , or marks it as retained in (the final) G_1^* and G_2^* by deleting some other unmarked units from G_1 and G_2 . In either case, a smaller instance is generated with the target solution value decreased by at least 1. The FPT algorithm recursively works on the smaller instances, with the terminating instances containing no isolates.

The FPT algorithm examines G_1 from head to tail (from left to right) to locate the first isolate, denoted as u . If there is none, then the algorithm terminates and it marks all unmarked units. Note that u might not be the first unmarked unit in G_1 , neither the first isolate in G_2 . Note also that u has to have an adjacent companion to stay together in a maximal common substring of G_1^* and G_2^* . Let P_1 denote the rightmost marked unit to the left of u in G_1 , if exists; let S_1 denote the leftmost marked unit to the right of u in G_1 , if exists. By Lemma 1, this companion has to be the last letter of P_1 , or the first letter of S_1 , or some unmarked unit in between P_1 and S_1 .

Lemma 2 *If isolate u is retained in the final G_1^* and G_2^* , then at least one unmarked unit must be deleted.*

PROOF. Let P_2 denote the rightmost marked unit to the left of u in G_2 , if exists; let S_2 denote the leftmost marked unit to the right of u in G_2 , if exists (Figure 1). We consider the following three cases.

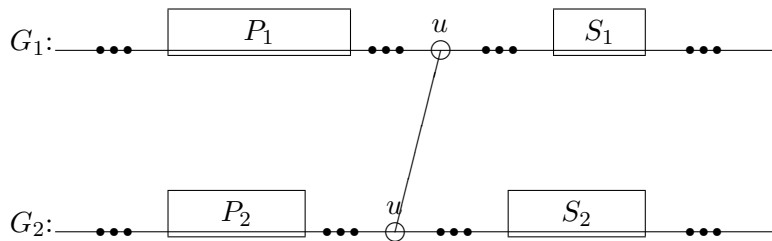


Figure 1: The local configuration at isolate u .

In the first case, the adjacent companion of isolate u to stay together in a maximal common substring of G_1^* and G_2^* is the last letter of P_1 . It follows that P_1 and P_2 are identical. If there are letters in between P_1 and u in G_1 , the lemma is proved. Otherwise, $P_1 \cdot u$ is a substring of G_1 . Similarly, if there are letters in between P_2 and u in G_2 , the lemma is proved. Otherwise, $P_2 \cdot u$ is a substring of G_2 . This is a contradiction to the assumption that u is an isolate.

In the second case, the adjacent companion of isolate u to stay together in a maximal common substring of G_1^* and G_2^* is the first letter of S_1 . This case can be similarly argued as in the first case.

In the last case, the adjacent companion of isolate u to stay together in a maximal common substring of G_1^* and G_2^* is some unmarked unit, denoted as V , in between P_1 and S_1 in G_1 . V has

to be in between P_2 and S_2 in G_2 . Assume without loss of generality that u appears to the left of V in G_1 . Then, u also appears to the left of V in G_2 (or $-u$ appears to the right of $-V$ in G_2). There must be some letters in between u and V in G_1 or in G_2 , otherwise contradicting to the assumption u is an isolate. These letters are not retained in G_1^* and G_2^* . \square

The key step in the FPT algorithm is to locate this adjacent companion of isolate u , if it is retained. The search process goes as follows. Compatible to the proof of Lemma 2, the algorithm scans from isolate u to the left in G_1 to locate the first unit V_1 that also appears to the left of u but not passing P_2 in G_2 , if exists. This V_1 can be an unmarked unit, can also be P_1 (and thus P_2), but not to the left of P_1 . The intention is that, if indeed the adjacent companion of isolate u as described in Lemma 2 is to the left of u , then either it is V_1 , or otherwise V_1 has to be deleted. Likewise, the algorithm scans from isolate u to the left in G_2 to locate the first unit V_2 that also appears to the left of u but not passing P_1 in G_1 , if exists. Similarly, this V_2 can be an unmarked unit, can also be P_2 (and thus P_1), but not to the left of P_2 .

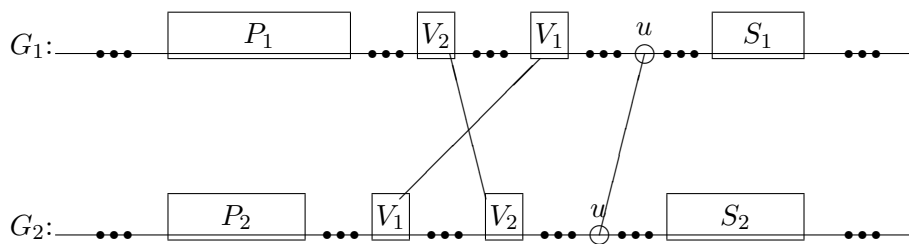


Figure 2: Searching for V_1 and V_2 to the left of isolate u .

Note that V_1 exists if and only if V_2 exists, since V_1 is a candidate when searching for V_2 and vice versus (Figure 2). In Case L1, V_1 and V_2 are identical; thus firstly there must be some unmarked units in between V_1 and u in either G_1 or G_2 ; and secondly the algorithm deletes these units to generate an instance of solution value $k' \leq k - 1$, since at least a letter is deleted. In Case L2, V_1 and V_2 are different; thus they are unmarked and they cannot co-exist in G_1^* and G_2^* ; let A_1 denote the set of letters in between V_1 and u (including V_1) in G_1 , and A_2 denote the set of letters in between V_2 and u (including V_2) in G_2 , then the algorithm deletes either all the letters of A_1 or all the letters of A_2 to generate two instances of solution value $k' \leq k - 2$, since $|A_1|, |A_2| \geq 2$.

The algorithm also needs to explore to the right of isolate u if the adjacent companion of isolate u as described in Lemma 2 is to the right of u , and we use W_1 and W_2 to denote the counterparts (Figure 3). First of all, if V_1/V_2 do not exist, then W_1/W_2 must exist for otherwise isolate u has to be deleted. In Case R1, W_1 and W_2 are identical; then similarly there must be some unmarked units in between u and W_1 in either G_1 or G_2 ; and the algorithm deletes these units to generate an instance of solution value $k' \leq k - 1$. In the other case, W_1 and W_2 are different; let B_1 denote the set of letters in between u and W_1 (including W_1) in G_1 , and B_2 denote the set of letters in between u and W_2 (including W_2) in G_2 .

If $|B_1| = 1$, then W_1 is an isolate; the algorithm proceeds to scan from isolate W_1 further to the right in G_1 to locate the first unmarked unit X_1 that also appears to the right of u in G_2 . Since

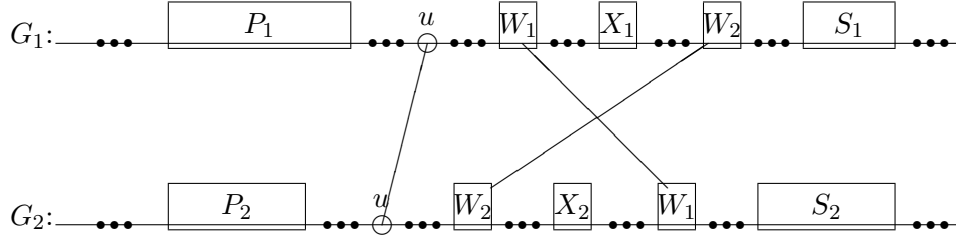


Figure 3: Searching for W_1 , W_2 , X_1 , and X_2 to the right of isolate u .

W_2 is a candidate for X_1 , we conclude that such X_1 exists in between W_1 and W_2 , or it is W_2 . In Case R2, X_1 sits to the left of W_1 in G_2 and there are no units in between W_1 and X_1 in G_1 ; the algorithm deletes W_1 to generate an instance of solution value $k' = k - 1$. The reason is that if W_1 is retained in G_1^* and G_2^* , one can always delete it and add back X_1 to have another optimal solution. In the other case, if there are no units in between W_1 and X_1 in G_1 , then X_1 must sit to the right of W_1 in G_2 , and the algorithm adds the letters of X_1 to B_1 ; if X_1 sits to the left of W_1 in G_2 , then there must be some units in between W_1 and X_1 in G_1 , and the algorithm adds these units into B_1 . By such an adding process, B_1 contains at least two letters.

If $|B_2| = 1$, the algorithm does exactly the same procedure, either falling into Case R2 to delete W_2 and generate an instance of solution value $k' = k - 1$, or adding at least one extra letter to B_2 .

Consequently, in the remaining case (Case R3), the algorithm faces with two sets B_1 and B_2 , both of size at least 2. Due to the fact that none of the letters of B_1 can co-exist with any letter of B_2 , the algorithm deletes either all the letters of B_1 or all the letters of B_2 to generate two instances of solution value $k' \leq k - 2$, since $|B_1|, |B_2| \geq 2$.

Let $f(k)$ be the number of smaller instances that are solved for assembling an optimal solution to the instance of solution value k . The above analysis states that the following recurrence holds.

$$f(k) \leq \begin{cases} f(k-1), & \text{if } u \text{ is deleted;} \\ + & \max \begin{cases} O(1), & \text{if } u \text{ is kept and } V_1/V_2 \text{ do not exist;} \\ f(k-1), & \text{if } u \text{ is kept and falls into Case L1;} \\ 2f(k-2), & \text{if } u \text{ is kept and falls into Case L2;} \end{cases} \\ + & \max \begin{cases} O(1), & \text{if } u \text{ is kept, } V_1/V_2 \text{ exist, but } W_1/W_2 \text{ do not exist;} \\ f(k-1), & \text{if } u \text{ is kept and falls into Case R1;} \\ f(k-1), & \text{if } u \text{ is kept and falls into Case R2;} \\ 2f(k-2), & \text{if } u \text{ is kept and falls into Case R3.} \end{cases} \end{cases}$$

Solving this recurrence gives us $f(k) \leq 3^k$. Since the preprocessing to determine all maximal common substrings of G_1 and G_2 needs at most quadratic time, so does the scanning process for isolate u , V_1 , V_2 , W_1 , W_2 , X_1 and X_2 , if necessary and if they exist, the total running time of the FPT algorithm is $O(3^k n^2) = O^*(3^k)$. A high-level description of the algorithm is in Figure 4. Therefore, we have proved the following theorem.

Theorem 3 *The CMSR problem can be solved in $O(3^k n^2)$ time, where n is the number of gene markers, k is the minimum number of gene markers such that deleting them from the genomic maps partitions the remainder maps into a common set of synthetic blocks of size at least two.*

Input: two sequences (permutations) G_1 and G_2 on n letters, parameter k .
Output: a set of k letters or less, removing which from the two sequences results in a partition into maximal common substrings of length at least 2.

1. Determines all type-0 substrings, and marks those of length 4 and greater;
2. Scans G_1 from head to tail (left to right) for the first isolate u ;
 - 2.1. if no such u found, return;
 - 2.2. sets “flag off”;
3. Deletes u , and recursively calls on the remainder sequences with parameter $k - 1$;
4. Scans in G_1 from u to its left for V_1 ;
 - 4.1. if no such V_1 found, sets flag on;
 - 4.2. else scans in G_2 from u to its left for V_2 ;
 - 4.2.1. if $V_1 = V_2$, executes Case L1;
 - 4.2.2. else executes Case L2;
5. Scans in G_1 from u to its right for W_1 ;
 - 5.1. if no such W_1 found and flag on, return false;
 - 5.2. else if no such W_1 found, return;
 - 5.3. else scans in G_2 from u to its right for W_2 ;
 - 5.3.1. if $W_1 = W_2$, executes Case R1;
 - 5.3.2. else
 - 5.3.2.1. lets B_1 be the set of letters from u up to W_1 in G_1 , including W_1 ;
lets B_2 be the set of letters from u up to W_2 in G_2 , including W_2 ;
 - 5.3.2.2. if $|B_1| = 1$, scans in G_1 from W_1 to its right for X_1 ;
 - 5.3.2.2.1. if no units in between W_1 and X_1 in G_1 and X_1 is to the left of W_1 in G_2 ,
executes Case R2;
 - 5.3.2.2.2. else if no units in between W_1 and X_1 in G_1 ,
then adds the letters of X_1 to B_1 ;
 - 5.3.2.2.3. else if X_1 is to the left of W_1 in G_2 ,
then adds the letters in between W_1 and X_1 in G_1 to B_1 ;
 - 5.3.2.3. if $|B_2| = 1$, scans in G_2 from W_2 to its right for X_2 ;
 - 5.3.2.3.1. if no units in between W_2 and X_2 in G_2 and X_2 is to the left of W_2 in G_1 ,
executes Case R2;
 - 5.3.2.3.2. else if no units in between W_2 and X_2 in G_2 ,
then adds the letters of X_2 to B_2 ;
 - 5.3.2.3.3. else if X_2 is to the left of W_2 in G_1 ,
then adds the letters in between W_2 and X_2 in G_2 to B_2 ;
 - 5.3.2.4. executes Case R3;

Figure 4: A high-level description of the FPT algorithm for the CMSR problem.

4 A 3-approximation algorithm

From Lemma 1, all type-0 substrings of length 4 and greater are retained in our approximation algorithm to be presented next. The output of our algorithm will be compared against an optimal solution OPT which also retains all these substrings. In the following, we only deal with length-3 and length-2 type-0 substrings, and isolates.

In the first step, our algorithm retains all length-3 and length-2 type-0 substrings. In the second step, our algorithm recursively removes one isolate; such a removed isolate has to satisfy the condition (C) listed in the following, with the goal that removing it from (the current) G_1 and G_2 gives rise to (at least) a new common substring of length 2. This new common substring is not a common substring to the original G_1 and G_2 , and is called a type-1 substring for distinction purpose. Note that after such isolate removal, some units (type-0 and/or type-1 substrings, and/or isolates) might be able to be merged into longer maximal common substrings. For consistency purpose, we do not merge two existing substrings; but we will append isolates to existing substrings (type-0 or type-1) whenever possible, since our goal is to get rid of isolates. These appended isolates become no longer isolates, and the extended substrings keep their type (type-0 or type-1).

The isolate chosen to be removed by our algorithm has to satisfy the following condition (C). When none can be identified, the algorithm enters the last step to remove all the remaining isolates, if any. Denote this target isolate as u .

- (C) In either G_1 or G_2 , two neighboring units of u are also isolates; and after removing u , they form into a type-1 common substring of length 2.

It could be the case that in both G_1 and G_2 , the two neighboring units of u form into a type-1 common substring of length 2 after deleting u ; our algorithm will identify the case and subsequently all these isolates become no longer isolates. There is another (disjoint) case in which, besides forming the type-1 common substring of length 2, another neighboring isolate of u in different sequence can be appended to an existing, or the newly formed, substring; our algorithm will identify this case too and subsequently the appended isolate becomes no longer an isolate. Intuitively, removing isolate u saves (*i.e.*, retains) at least two other isolates, and can save one or two more isolates.

For ease of discussion, let $U = \{u_1, u_2, \dots, u_m\}$ denote the set of isolates located in sequential order by our algorithm, which are all removed. Associated with each u_j , let V_j denote the set of neighboring isolates of u_j in the current G_1 and G_2 that become no longer isolates after removing u_j . We have $|V_j| \geq 2$, for $j = 1, 2, \dots, m$. In particular, the two neighboring isolates of u_j that form a type-1 substring after deleting u_j are denoted as a_j and b_j (where there are two such pairs, a_j and b_j refer to an arbitrary one of them). Let R denote the set of remaining isolates at the time the algorithm finds no isolates satisfying condition (C); that is, R is the set of isolates deleted by our algorithm at the last step. The following two lemmas state some preliminary observations.

Lemma 4 *The set of all isolates I is the union of the disjoint sets U, V_1, V_2, \dots, V_m , and R , that is, $I = U \cup \left(\bigcup_{j=1}^m V_j\right) \cup R$; moreover, the algorithm deletes all isolates of $U \cup R$, but no others.*

Lemma 5 *In the original input sequences G_1 and G_2 , the letters in between a_j and b_j all belong to $\{u_1, u_2, \dots, u_{j-1}, u_j\}$; moreover, u_j is in between a_j and b_j in exactly one of G_1 and G_2 .*

Recall that we use in the discussion an optimal solution OPT which satisfies the two properties listed in Lemma 1. We partition OPT into a subset O_3 of length-3 type-0 substrings, a subset O_2 of length-2 type-0 substrings, and a subset O_1 of isolates: $OPT = O_3 \cup O_2 \cup O_1$. These substrings and isolates are referred to as units in the sequel. Consider the inverse process of deleting units of OPT from G_1 and G_2 to obtain the final sequences G_1^* and G_2^* . In this inverse process, we add the units of OPT back to G_1^* and G_2^* using their original positions in G_1 and G_2 to re-construct G_1 and G_2 . At the beginning of this process, there are no isolated letters in G_1^* or G_2^* ; all the isolates of I are thus either units of $I \cap O_1$, or *generated* by inserting units of OPT back, which break the maximal common substrings into fragments of which some are single letters. At any time of the process, inserting one unit of OPT back to the current G_1 and G_2 can generate at most four fragments of single letters, since in the worst case two current length-2 substrings can be broken into four such fragments. Some of these single letters might not be the isolates of $U \cup R$; those that are in $U \cup R$, as well as the inserted unit when it belongs to $(U \cup R) \cap O_1$, are said to be associated with the inserted unit of OPT . We firstly insert units of O_3 and O_2 , one by one; each of them is associated with at most four isolates of $U \cup R$ (Lemma 6); the resultant sequences are denoted as G_1^0 and G_2^0 .

Lemma 6 *The number of isolates of $U \cup R$ associated with each unit of $O_3 \cup O_2$ is at most four.*

Next, we insert isolates of $O_1 \cap (u_j \cup V_j)$ back into G_1^0 and G_2^0 , for $j = 1, 2, \dots, m$ sequentially. At the end of the inserting isolates of $O_1 \cap (u_j \cup V_j)$, the resultant sequences are denoted as G_1^j and G_2^j . We emphasize that this sequential order is very important, as we need it in the proof of Lemma 8, which counts the average number of isolates of $U \cup R$ associated with each isolate of $O_1 \cap (u_j \cup V_j)$.

Lemma 7 *For any j , u_j is an isolated letter in G_1^j and G_2^j .*

PROOF. We prove this lemma by (finite) induction. Firstly, we notice that a_1, b_1 , and u_1 cannot co-exist in G_1^* and G_2^* , since otherwise u_1 would be the only letter in between a_1 and b_1 in exactly one of G_1^* and G_2^* , and thus an isolated letter. Therefore, $O_1 \cap (u_1 \cup V_1) \neq \emptyset$. After inserting isolates of $O_1 \cap (u_1 \cup V_1)$ back, a_1, b_1 , and u_1 are all present in G_1^1 and G_2^1 . For the same reason that u_1 is the only letter in between a_1 and b_1 in exactly one of G_1^1 and G_2^1 , u_1 is an isolated letter. That is, the lemma holds for $j = 1$.

Assume the lemma holds for all $i = 1, 2, \dots, j - 1$, that is, u_1, u_2, \dots, u_{j-1} are isolated letters in G_1^{j-1} and G_2^{j-1} , and thus they are all isolated letters in G_1^j and G_2^j . Due to the co-existence of a_j, b_j , and u_j in G_1^j and G_2^j , Lemma 5 tells that if u_j is not an isolated letter, then it can only pair with some letter of $\{u_1, u_2, \dots, u_{j-1}\}$ to sit together in a substring. This is a contradiction to the inductive assumption. Therefore, u_j is an isolated letter in G_1^j and G_2^j . \square

Lemma 8 *For any j , the average number of isolates of $U \cup R$ associated with isolates of $O_1 \cap (u_j \cup V_j)$ is at most 3. Moreover, by the end of this iteration of inserting process, u_j is associated to some unit of OPT .*

PROOF. Recall that we insert isolates of $O_1 \cap (u_j \cup V_j)$ back into G_1^0 and G_2^0 in sequential order of j . When we start to insert isolates of $O_1 \cap (u_j \cup V_j)$, all isolates of $O_1 \cap \left(\bigcup_{i=1}^{j-1} u_i \cup V_i\right)$ have been inserted and the resultant sequences are G_1^{j-1} and G_2^{j-1} .

Firstly, if $O_1 \cap (u_j \cup V_j) = \emptyset$, then by Lemma 7 we know u_j is an isolated letter in G_1^{j-1} and G_2^{j-1} . That is, u_j was generated by the earlier inserting process. We assume in the following that $O_1 \cap (u_j \cup V_j) \neq \emptyset$. We want to associate u_j , among others, to isolates of $O_1 \cap (u_j \cup V_j)$, if u_j was not previously generated as an isolated letter. This way, we guarantee that u_j is associated to units of OPT by the end of this iteration of inserting process. Let a_j and b_j be the two neighboring isolates of u_j when the approximation algorithm located u_j , as in Lemma 5, such that by removing u_j , $a_j \cdot b_j$ became a type-1 length-2 substring. We consider the following two disjoint cases: $u_j \in O_1$ and $u_j \notin O_1$.

In the first case, $u_j \in O_1$. When $a_j, b_j \in O_1$ and a_j and b_j are separated by certain letters of $\{u_1, u_2, \dots, u_{j-1}\}$ in G_1 (G_2 , respectively), inserting a_j and b_j into G_1^{j-1} (G_2^{j-1} , respectively) does not generate any new isolates of $U \cup R$; when $a_j, b_j \in O_1$ and a_j and b_j are separated by no letters of $\{u_1, u_2, \dots, u_{j-1}\}$ in G_1 (G_2 , respectively), inserting a_j and b_j into G_1^{j-1} (G_2^{j-1} , respectively) can generate at most two isolates of $U \cup R$. When one and only one of a_j and b_j is in O_1 , then inserting it into G_1^{j-1} and G_2^{j-1} does not generate any new isolates of $U \cup R$.

If $|V_j| = 4$, then the other two letters, c_j and d_j , have the same properties as a_j and b_j . When $|V_j \cap O_1| = 4$, that is, $a_j, b_j, c_j, d_j \in OPT$, inserting a_j, b_j and c_j, d_j can generate at most 8 new isolates of $U \cup R$. When $|V_j \cap O_1| = 3$, and assuming $a_j, b_j, c_j \in OPT$, inserting a_j, b_j can generate at most 4 new isolates of $U \cup R$, but inserting c_j generates no new isolates of $U \cup R$. When $|V_j \cap O_1| = 2$, and in the first scenario assuming $a_j, b_j \in OPT$, inserting a_j, b_j can generate at most 4 new isolates of $U \cup R$; in the second scenario assuming $a_j, c_j \in OPT$, inserting a_j, c_j generates no new isolates of $U \cup R$. When $|V_j \cap O_1| = 1$, and assuming $a_j \in OPT$, inserting a_j generates no new isolates of $U \cup R$. After inserting isolates of $O_1 \cap V_j$, if any, inserting u_j back into the current G_1^{j-1} and G_2^{j-1} does not generate any new isolates of $U \cup R$, but u_j itself. In summary, for $|O_1 \cap V_j| = 4, 3, 2, 1$, and 0, respectively, the total number of isolates of $U \cup R$ associated with isolates of $O_1 \cap (u_j \cup V_j)$ is at most 9, 5, 5, 1, and 1, respectively. It follows that the average number of isolates of $U \cup R$ associated with isolates of $O_1 \cap (u_j \cup V_j)$ is at most $9/5$.

If $|V_j| = 3$, then the third letter, c_j , was appended to an existing (type-0 or type-1) substring S when the approximation algorithm removed u_j . Similarly to the discussion on a_j and b_j , c_j and S can only be separated by letters of $\{u_1, u_2, \dots, u_{j-1}\}$, besides u_j , in G_1 and G_2 . Moreover, u_j is in between c_j and S in at most one of G_1 and G_2 . Therefore, when $c_j \in O_1$, inserting it into G_1^{j-1} and G_2^{j-1} does not generate any new isolates of $U \cup R$. After inserting isolates of $O_1 \cap V_j$, if any, inserting u_j back into the current G_1^{j-1} and G_2^{j-1} does not generate any new isolates of $U \cup R$, but u_j itself. Therefore, for $|O_1 \cap V_j| = 3, 2, 1$, and 0, respectively, the total number of isolates of $U \cup R$ associated with isolates of $O_1 \cap (u_j \cup V_j)$ is at most 5, 5, 1, and 1, respectively. It follows that the average number of isolates of $U \cup R$ associated with isolates of $O_1 \cap (u_j \cup V_j)$ is at most $5/3$.

If $|V_j| = 2$, after inserting isolates of $O_1 \cap V_j$, if any, inserting u_j back into the current G_1^{j-1} and G_2^{j-1} can generate at most two isolates of $U \cup R$, besides u_j . Therefore, for $|O_1 \cap V_j| = 2, 1$, and 0, respectively, the total number of isolates of $U \cup R$ associated with isolates of $O_1 \cap (u_j \cup V_j)$ is at most 7, 3, and 3, respectively. It follows that the average number of isolates of $U \cup R$ associated with isolates of $O_1 \cap (u_j \cup V_j)$ is at most 3.

In the second case, $u_j \notin O_1$. Assume without loss of generality that u_j is in between a_j and b_j in G_1 in Lemma 5. When $a_j \in O_1$ ($b_j \in O_1$, respectively) and a_j (b_j , respectively) and u_j are separated by certain letters of $\{u_1, u_2, \dots, u_{j-1}\}$ in G_1 , inserting a_j (b_j , respectively) into G_1^{j-1} does not generate any new isolates of $U \cup R$ in G_1 . When $a_j \in O_1$ ($b_j \in O_1$, respectively) and a_j (b_j , respectively) and u_j are separated by no letters of $\{u_1, u_2, \dots, u_{j-1}\}$ in G_1 , inserting a_j (b_j , respectively) into G_1^{j-1} can generate at most two isolates of $U \cup R$ in G_1 , including u_j . Nonetheless, when $a_j, b_j \in O_1$ and a_j and b_j are separated by no letters of $\{u_1, u_2, \dots, u_{j-1}\}$ in G_1 , inserting a_j and b_j into G_1^{j-1} can generate at most three isolates of $U \cup R$ in G_1 , including u_j . Similarly, when $a_j, b_j \in O_1$ and a_j and b_j are separated by certain letters of $\{u_1, u_2, \dots, u_{j-1}\}$ in G_2 , inserting a_j and b_j into G_2^{j-1} does not generate any new isolates of $U \cup R$ in G_2 ; when $a_j, b_j \in O_1$ and a_j and b_j are separated by no letters of $\{u_1, u_2, \dots, u_{j-1}\}$ in G_2 , inserting a_j and b_j into G_2^{j-1} can generate at most two isolates of $U \cup R$ in G_2 .

If $|V_j| = 4$, then the other two letters, c_j and d_j , have the same properties as a_j and b_j . Note that when inserting a_j and b_j into G_1^{j-1} generates new isolates of $U \cup R$, these isolates will be seen again when inserting c_j and d_j into G_2^{j-1} . Therefore, for $|O_1 \cap V_j| = 4, 3, 2$, and 1, respectively, the total number of isolates of $U \cup R$ associated with isolates of $O_1 \cap (u_j \cup V_j)$ is at most 7, 4, 2, and 0, respectively. It follows that the average number of isolates of $U \cup R$ associated with isolates of $O_1 \cap (u_j \cup V_j)$ is at most $7/4$.

If $|V_j| = 3$, then the third letter, c_j , was appended to an existing (type-0 or type-1) substring S when the approximation algorithm removed u_j . Similarly to the discussion on a_j and b_j , c_j and S can only be separated by letters of $\{u_1, u_2, \dots, u_{j-1}\}$ in G_1 and G_2 , besides u_j in G_2 . Therefore, when $c_j \in O_1$, inserting c_j into G_2^{j-1} can generate at most one new isolate of $U \cup R$, which is u_j itself. Note that S might start with a_j or end with b_j . For $|O_1 \cap V_j| = 3, 2$, and 1, respectively, the total number of isolates of $U \cup R$ associated with isolates of $O_1 \cap (u_j \cup V_j)$ is at most 4, 2, and 0, respectively. It follows that the average number of isolates of $U \cup R$ associated with isolates of $O_1 \cap (u_j \cup V_j)$ is at most $4/3$.

If $|V_j| = 2$, for $|O_1 \cap V_j| = 2$, and 1, respectively, the total number of isolates of $U \cup R$ associated with isolates of $O_1 \cap (u_j \cup V_j)$ is at most 5, and 2, respectively. It follows that the average number of isolates of $U \cup R$ associated with isolates of $O_1 \cap (u_j \cup V_j)$ is at most $5/2$.

From the above case analysis, we conclude that the average number of isolates of $U \cup R$, which include u_j , associated with isolates of $O_1 \cap (u_j \cup V_j)$ in the worst case is 3. This finishes the proof. \square

Lastly, we insert isolates of $O_1 \cap R$ back into G_1^m and G_2^m . At the end of this last inserting process, we achieve the input sequences G_1 and G_2 .

Lemma 9 *The average number of isolates of $U \cup R$ associated with isolates of $O_1 \cap R$ is at most 3.*

PROOF. The key fact used in the proof is that after locating isolate u_m , removing it from the current sequences, and making letters in V_m non-isolates, the approximation algorithm finds no more isolates to iterate the process. That is, for any two remaining isolates $r, s \in R$ that are not separated by any existing (type-0 or type-1) substring in both sequences (that is, r and s can

potentially form into a substring, or participate together), there are at least two other isolates, duplications are separately counted, in between them, counting from both sequences.

In sequences G_1^m and G_2^m obtained after inserting units of $O_3 \cup O_2 \cup (O_1 \cap (U \cup \cup_{j=1}^m V_j))$ into G_1^* and G_2^* , all isolates of U are isolated letters and they have been associated to some units of OPT respectively by Lemma 8. For units of R , some are already isolated letters and they have been associated to some units of OPT respectively, while the others still reside in substrings (of length at least two). These units still residing in substrings are to be singled out by inserting units of $O_1 \cap R$ into G_1^m and G_2^m ; and it is these units that are associated with isolates of $O_1 \cap R$. We estimate next how many of these units there are.

Let S_1, S_2, \dots, S_k denote the substrings in G_1^m and G_2^m that are made of isolates of R ; and T_1, T_2, \dots, T_ℓ denote the fragments of substrings in G_1^m and G_2^m , where the substrings are not purely made of isolates of R , but the fragments are. Note that $|S_i| \geq 2$ for every i . To single out all letters of $(\cup_{i=1}^k S_i) \cup (\cup_{j=1}^\ell T_j)$, we first need at least one isolate of $O_1 \cap R$ to chop each T_i off its host substring; afterwards, the above argument states that for every two adjacent letters in S_i or T_j , there are at least two isolates of $O_1 \cap R$ in between them, counting from both sequences. This gives a lower bound on the minimum number of isolates of $O_1 \cap R$. Since each isolate of $O_1 \cap R$ can appear in two places, we have

$$2|O_1 \cap R| \geq \ell + \sum_{i=1}^k 2(|S_i| - 1) + \sum_{j=1}^\ell 2(|T_j| - 1) \geq \sum_{i=1}^k |S_i| + \sum_{j=1}^\ell |T_j|.$$

Therefore, the total number of isolates of $U \cup R$ (in this case, R only) that are associated with isolates of $O_1 \cap R$ is at most $\sum_{i=1}^k |S_i| + \sum_{j=1}^\ell |T_j| + |O_1 \cap R|$, which is less than or equal to $3|O_1 \cap R|$. This proves the lemma. \square

Theorem 10 *The CMSR problem admits a 3-approximation algorithm.*

PROOF. To summarize, all isolates of $U \cup R$ are associated with units of OPT . From Lemmas 6, 8, and 9, we have

$$|U \cup R| \leq 4|O_3 \cup O_2| + 3|O_1 \cap (U \cup (\cup_{j=1}^m V_j))| + 3|O_1 \cap R| \leq \frac{4}{3} \times 3|O_3| + 2 \times 2|O_2| + 3 \times |O_1| \leq 3|OPT|,$$

where $|OPT|$ denotes the number of letters in OPT and thus $|OPT| = 3|O_3| + 2|O_2| + |O_1|$. Note that the algorithm deletes all isolates of $U \cup R$, but no others, and therefore it is a 3-approximation algorithm. \square

5 Conclusions

In this paper, we presented a fixed-parameter tractable algorithm and a 3-approximation algorithm for the CMSR problem. The running time of the FPT algorithm is $O^*(3^k)$, where k is the size of the optimal solutions. We believe that a more careful analysis on the local configuration of an isolate can lead to faster FPT algorithms. In the approximation algorithm, the key design technique is greedy, and the performance ratio is proven using an inverse amortized analysis. Better approximation algorithms are certainly our future work.

Acknowledgment

We thank Henning Fernau for his valuable comments, and anonymous reviewers for several insightful comments on an earlier version of the paper. HJ and BZ are partially supported by NSF grant DMS-0918034 and NSF of China under project 60928006. ZL and GL are supported by NSERC. LW is fully supported by a grant from City University of Hong Kong (Project No. 7002452).

References

- [1] R. Bar-Yehuda, M. M. Halldórsson, J. S. Naor, H. Shachnai, and I. Shapira. Scheduling split intervals. *SIAM Journal on Computing*, 36:1–15, 2006.
- [2] L. Bulteau, G. Fertin and I. Rusu. Maximal strip recovery problem with gaps: hardness and approximation algorithms. *Proceedings of the 20th Annual International Symposium on Algorithms and Computation (ISAAC'09)*, LNCS 5878, pages 710–719, 2009.
- [3] Z. Chen, B. Fu, M. Jiang, and B. Zhu. On recovering synthetic blocks from comparative maps. *Journal of Combinatorial Optimization*, 18:307–318, 2009.
- [4] V. Choi, C. Zheng, Q. Zhu, and D. Sankoff. Algorithms for the extraction of syntheny blocks from comparative maps. In *Proceedings of the 7th International Workshop on Algorithms in Bioinformatics (WABI'07)*, pages 277–288, 2007.
- [5] R. Downey and M. Fellows. *Parameterized Complexity*, Springer-Verlag, 1999.
- [6] M. Jiang. Inapproximability of maximal strip recovery. *Proceedings of the 20th Annual International Symposium on Algorithms and Computation (ISAAC'09)*, LNCS 5878, pages 616–625, 2009.
- [7] M. Jiang. Inapproximability of maximal strip recovery, II. *Proceedings of the 4th Annual Frontiers of Algorithmics Workshop (FAW'10)*, LNCS 6213, pages 53–64, 2010.
- [8] L. Wang and B. Zhu. On the tractability of maximal strip recovery. *J. of Computational Biology*, 17(7):907–914, 2010. (A one-page correction is to appear in January 2011.)
- [9] C. Zheng, Q. Zhu, and D. Sankoff. Removing noise and ambiguities from comparative maps in rearrangement analysis. *IEEE/ACM Transactions on Computational Biology and Bioinformatics*, 4:515–522, 2007.