Journal of Computer and System Sciences 78 (2012) 720–730

Contents lists available at SciVerse ScienceDirect

Journal of Computer and System Sciences

www.elsevier.com/locate/jcss

An improved approximation algorithm for the complementary maximal strip recovery problem

Guohui Lin [a,*], Randy Goebel [a], Zhong Li [a], Lusheng Wang [b]

[a] Department of Computing Science, University of Alberta, Edmonton, Alberta T6G 2E8, Canada
[b] Department of Computer Science, City University of Hong Kong, Kowloon, Hong Kong, China

ARTICLE INFO

Article history:
Received 14 May 2011
Received in revised form 3 October 2011
Accepted 21 October 2011
Available online 29 October 2011

Keywords:
Maximal strip recovery
Approximation algorithm
Local amortized analysis
Re-weighting scheme

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 strips, which are common substrings of length greater than or equal to two. The *complementary* maximal strip recovery (CMSR) problem has the complementary goal to delete the minimum number of markers. Both MSR and CMSR have been shown to be NP-hard and APX-complete, and they admit a 4-approximation and a 3-approximation respectively. In this paper, we present an improved $\frac{7}{3}$-approximation algorithm for the CMSR problem, with its worst-case performance analysis done through a *local amortization* with a *re-weighting scheme*.

© 2011 Elsevier Inc. All rights reserved.

1. Introduction

In comparative genomics, one of the first steps is to decompose two given genomes into syntenic blocks — segments of chromosomes that are deemed homologous in the two input genomes. Many decomposition methods have been proposed, but they are vulnerable to ambiguities and errors, which are isolated points that do not co-exist with other points in genomic maps [4,9]. The *maximal strip recovery* (MSR) problem was formulated for eliminating these noise and ambiguities. In a more precise formulation, we are given two genomic maps G_1 and G_2 each represented as a sequence of n distinct gene markers (which form the alphabet Σ), 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 strips, which are common 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 to be NP-hard [8,2,3]. More recently, it is shown to be APX-complete [6], admitting a 4-approximation algorithm [3]. This 4-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 recently proven to be APX-hard [7], admitting a 3-approximation algorithm [5]. Our main result is an improved $\frac{7}{3}$-approximation algorithm for CMSR. As we will show later, the key design technique is a local greedy scheme, on top of six operations of three priority levels, to retain

* Corresponding author.
E-mail addresses: guohui@ualberta.ca (G. Lin), rgoebel@ualberta.ca (R. Goebel), zhong4@ualberta.ca (Z. Li), cswangl@cityu.edu.hk (L. Wang).

0022-0000/$ – see front matter © 2011 Elsevier Inc. All rights reserved.
doi:10.1016/j.jcss.2011.10.014

certain isolates while deleting some other ones; and the performance ratio is proven using a novel technique called *local amortized analysis* with a *re-weighting scheme*. Some preliminary ideas of the algorithm design and analysis have appeared in [5], where the first constant-ratio approximation algorithm is presented.

2. Preliminaries

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 reversed and negated form. We reserve the bullet symbol "•" for connection use, for example, $a \bullet b$ in sequence G_1 means gene b comes directly after gene a in G_1. When a common substring (also called *strip*, or *syntenic block*) of the two current target sequences (*i.e.*, G_1 and G_2, or their remainders after deleting some letters) is specified, it is of length greater than or equal to two, unless otherwise explicitly stated that it is a single letter; The substring will (often) be labeled using a capital letter. We abuse this capital letter a little bit to also denote the set of gene markers in the substring, when there is no ambiguity. We present several important structural properties of the CMSR problem in this section, which are used in the design of the approximation algorithm and its performance analysis in the next section.

We first look at a warm-up instance. In this instance, $G_1 = \langle a, b, c, d, e, f, g, h, i, j, k, \ell\rangle$ and $G_2 = \langle -i, -d, -g, -f, h, a, c, b, -\ell, -k, -j, -e\rangle$ (commas are used to separate the gene markers for easier reading). There are a length-2 maximal common substring $f \bullet g$ (appearing in the reversed and negated form in G_2) and a length-3 maximal common substring $j \bullet k \bullet \ell$ (appearing in the reversed and negated form in G_2), while the other letters do not form into common substrings. If we delete letter c from both sequences, then another length-2 maximal common substring $a \bullet b$ can be formed. Furthermore, by deleting markers d, e, and h from both G_1 and G_2, the remainder sequences 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 remainder sequences can be decomposed into three maximal common substrings $a \bullet b$, $f \bullet g \bullet i$, and $j \bullet k \bullet \ell$. 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.

In the rest of the paper, we use *OPT* to denote an optimal solution to the instance of the CMSR problem. That is, *OPT* is a minimum-size subset of letters that, deleting them from G_1 and G_2 gives the remainder sequences, denoted G_1^* and G_2^*, respectively, which can be partitioned into maximal common substrings.

For the CMSR instance, in at most quadratic time, we can determine all maximal common substrings of G_1 and G_2 (of length at least two) and the isolated letters that do not belong to any of the common substrings. We use *unit* to refer to a maximal common substring or an isolated letter. A unit and its reversed and negated form are considered identical. The units determined above form a common partition of G_1 and G_2, *i.e.*, every letter occurs in exactly one of these units. For ease of presentation, these maximal common substrings are called *type*-0 substrings; the isolated letters are called *isolates*. In our algorithm Approx-CMSR to be presented in the next section, all the type-0 substrings are kept in the final sequences and our goal is to eliminate the isolates, by either deleting them from the input sequences, or to "*merge*" them into substrings right after deleting some other isolates from the input sequences. Here "merging" refers to either appending an isolate to some existing substring, or forming two isolates into a novel common substring, which is called a *type*-1 substring. That is, our focus is on the isolates. Note that every isolate appears exactly once in each of the two sequences G_1 and G_2; for distinction purpose we use an *isolate copy* to refer to the isolate in one of the two sequences, while reserve *isolate* to refer to both its copies as a whole.

Lemma 1. *(See [5].) For any CMSR instance, there exists an optimal solution OPT such that*

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

The above Lemma 1 shows that in the optimal solution, for every type-0 substring, either all its letters are deleted or none of them is deleted. We partition *OPT* into a subset O_3 of letters residing in the length-3 type-0 substrings, a subset O_2 of letters residing in the length-2 type-0 substrings, and a subset O_1 of isolates: $OPT = O_3 \cup O_2 \cup O_1$. Recall that the type-0 substrings and isolates deleted in the optimal solution are referred to as *units* of *OPT*.

2.1. Favorable operations

In our algorithm Approx-CMSR to be presented in the next section, all the type-0 substrings are kept in the final sequences. The letters whose decision needs to be made are the isolates. Let a and b be two arbitrary isolates. They can form into a common substring after the letters in between them in the two sequences are all deleted. Therefore, we consider only such an isolate pair that the letters in between them in the two sequences are all isolates. If there is only one isolate in between a and b, say x, in the two sequences, then deleting x to merge a and b into a length-2 substring has a *gain* of at least one, *i.e.*, deleting one but keeping at least two letters. We distinguish two scenarios. In the first scenario both copies of x are in between a and b, one in each of the two sequences. That is, $a \bullet x \bullet b$ appears in one of G_1 and G_2 and $a \bullet -x \bullet b$ appears in the other (see row 1 in Table 1, or $a \bullet x \bullet b$ appears in one of G_1 and G_2 and $-b \bullet x \bullet -a$ appears in the other, which can be analogously discussed). Deleting x and the subsequent merging is referred to as an operation 1, which is given a *high* priority.

Table 1
Six different isolate elimination operations with three levels of priorities. In these operations, lower case letters a, b, x, y, s, t, w, z are isolates, while S is an existing substring at the time of consideration. Assuming the operation is executed in the j-th iteration, U_j is the set of isolates deleted and V_j is the set of isolates kept by algorithm Approx-CMSR.

Priority	Operation	Local configuration	Sets	Re-weighting
High	1	$\ldots a \bullet x \bullet b \ldots$	$U_j = \{x\}$;	$x : \frac{1}{3}$;
		$\ldots a \bullet -x \bullet b \ldots$	$V_j = \{a, b\}$	$a, b : \frac{1}{3}$
	2	$\ldots s \bullet a \bullet x \bullet b \bullet t \ldots w \bullet z \ldots$	$U_j = \{a, b\}$;	$a, b : \frac{1}{3}$;
		$\ldots s \bullet x \bullet t \ldots w \bullet a \bullet b \bullet z \ldots$	$V_j = \{x, s, t, w, z\}$	$s, t, w, z : \frac{1}{3}$
Medium	3	$\ldots a \bullet x \bullet b \ldots$	$U_j = \{x\}$;	$x : \frac{1}{3}$;
		$\ldots a \bullet b \ldots$	$V_j \supseteq \{a, b\}$	$a, b : \frac{1}{3}$
Low	4	$\ldots a \bullet x \bullet y \bullet b \ldots$	$U_j = \{x, y\}$;	$x, y : \frac{2}{3}$;
		$\ldots a \bullet b \ldots$	$V_j \supseteq \{a, b\}$	$a, b : \frac{1}{3}$
	5	$\ldots a \bullet x \bullet b \ldots$	$U_j = \{x, y\}$;	$x, y : \frac{2}{3}$;
		$\ldots a \bullet y \bullet b \ldots$	$V_j \supseteq \{a, b\}$	$a, b : \frac{1}{3}$
	6	$\ldots a \bullet x \bullet S \ldots$	$U_j = \{x\}$;	$x : \frac{2}{3}$;
		$\ldots a \bullet S \ldots$	$V_j \supseteq \{a\}$	$a : \frac{1}{3}$

In the second scenario only one copy of x is in between a and b, in one of the two sequences. That is, $a \bullet x \bullet b$ appears in one of G_1 and G_2 and $a \bullet b$ appears in the other (see rows 2 and 3 in Table 1). Deleting x enables the merging of a and b into a novel substring $a \bullet b$. However, if deleting a and b can merge all their five neighboring isolates into substrings, see row 2 in Table 1, algorithm Approx-CMSR chooses to delete a and b. Deleting a and b and the subsequent merging is referred to as an operation 2, which is also given a high priority. In the other case (see row 3 in Table 1), algorithm Approx-CMSR deletes x and merges a and b into a substring, referred to as an operation 3, which is given a medium (*i.e.*, lower than high) priority. Note that in an operation 3, the two neighboring letters of the other copy of x (in the sequence where $a \bullet b$ appears), might also be merged together when one or both of them are isolates. It follows that the pure gain of an operation 3, though with only a medium priority, can be as large as 3. In all operations including three more to be introduced next, the isolates that are merged into substrings lose their isolate identities, and are kept by algorithm Approx-CMSR.

If merging a and b has to delete exactly two isolate copies of two distinct isolates, say a copy of x and a copy of y, the operation will have a low priority. We again distinguish two scenarios. In the first scenario both isolate copies are in between a and b in one of the two sequences. That is, $a \bullet x \bullet y \bullet b$ appears in one of G_1 and G_2 and $a \bullet b$ appears in the other (see row 4 in Table 1). In the second scenario one isolate copy is in between a and b in one of the two sequences while the other isolate copy is in between a and b in the other of the two sequences. That is, $a \bullet x \bullet b$ appears in one of G_1 and G_2 and $a \bullet y \bullet b$ appears in the other (see row 5 in Table 1). Note that the negated and reversed gene forms can be analogously discussed. Deleting x and y and the subsequent merging in the first scenario is referred to as an operation 4, and in the second scenario is referred to as an operation 5, respectively. Both operations have a gain of 0. It is important to notice that in both operations 4 and 5, when the two neighboring letters of the other copy of x (y, respectively) are both isolates, they should not be able to be merged due to the lower operation priority; additionally for an operation 4, if x and y can potentially be merged by deleting some other isolates, then the number of isolates in between the other copy of x and the other copy of y should be at least two, again due to the lower operation priority.

In an operation 4 or an operation 5, besides forming the substring $a \bullet b$, the two neighboring letters of the other copy of x (y, respectively) might be able to be merged, when exactly one is an isolate and the other one is in an existing substring. Nevertheless, the gain in this merging process is not counted towards the operation, as the algorithm does not care about it. Similarly, when none of an operation 1, or 2, or 3 can be executed, the algorithm also looks for a chance to append an isolate a to an existing substring S (of either type), which happens if there is exactly one isolate copy, say of x, in between a and S in the two sequences, that is, $a \bullet x \bullet S$ appears in one of G_1 and G_2 and $a \bullet S$ appears in the other (see row 6 in Table 1). Deleting x and the subsequent appending in this case is referred to as an operation 6, which has the same priority as operations 4 and 5. This means, when the two neighboring letters of the other copy of x are both isolates and they are able to be merged after deleting x, we should make it an operation 3 instead of an operation 6; nevertheless, in this operation 6, deleting x might be able to append another isolate to an existing substring.

Algorithm Approx-CMSR does not consider to merge two isolates a and b that are separated by more than two isolate copies, neither to append an isolate a to an existing substring S that are separated by more than one isolate copy, in the two sequences.

3. An improved approximation algorithm

We assume at hand an optimal solution *OPT* stated in Lemma 1, and it is partitioned as $OPT = O_3 \cup O_2 \cup O_1$.

Algorithm Approx-CMSR:

Input: two sequences (permutations) G_1 and G_2 on the same set of letters.
Output: two subsequences of G_1 and G_2 respectively, that can be partitioned into maximal common substrings of length at least 2.

1. Determines all type-0 substrings of G_1 and G_2, and retains them;
2. While (there are feasible operations in the current sequences G_1 and G_2), do
   2.1. finds an operation of the currently top priority;
   2.2. removes the letters of U_j from G_1 and G_2;
   2.3. retains the letters of V_j in G_1 and G_2 by forming appropriate substrings;
3. Deletes all the remaining isolates, the letters of R, from G_1 and G_2.

Fig. 1. A high-level description of algorithm Approx-CMSR.

3.1. The Approx-CMSR algorithm

In the first step of our approximation algorithm, denoted as Approx-CMSR, it retains all type-0 substrings. That is, Approx-CMSR will only delete isolates from the input sequences (in the second and the third steps).

In the second step, Approx-CMSR iteratively removes one or two isolates; the candidate isolates have to be in one of the six cases listed in Table 1, and the operation with the top priority is chosen (tie breaks arbitrarily) for execution at the time of consideration. In each of the six cases, the isolate removal can give rise to novel common substrings to the remainder sequences (all except operation 6), and/or allow an isolate to be appended to an existing common substring in the remainder sequences (all except operations 1 and 2). If a novel common substring is formed, it is referred to as a type-1 substring; if an existing substring is extended, it retains its type for ease of presentation. The involved isolates that are merged into substrings, in either case, lose their isolate identities and are retained by Approx-CMSR.

Let $\mathcal{U} = \{U_1, U_2, \ldots, U_m\}$, where U_j denotes the set of isolates deleted by Approx-CMSR in the j-th iteration of the second step. Correspondingly, let V_j denote the set of isolates that are retained by Approx-CMSR in the j-th iteration. In Table 1, the U_j and V_j for each of the six operations are (partially) specified. Let R denote the set of remaining isolates at the time the algorithm finds no operations to execute (at the end of the m-th iteration). In the last step of the algorithm, Approx-CMSR deletes all letters of R from the two sequences. A high-level description of the algorithm Approx-CMSR is depicted in Fig. 1.

3.2. Performance analysis

Let $U = \bigcup_{j=1}^{m} U_j$ and $V = \bigcup_{j=1}^{m} V_j$. The following two lemmas state some preliminary observations on algorithm Approx-CMSR.

Lemma 2. *The set of all isolates is the union of the disjoint sets $U_1, U_2, \ldots, U_m$, $V_1, V_2, \ldots, V_m$, and R, that is, $U \cup V \cup R$; Algorithm* Approx-CMSR *deletes exactly all isolates of $U \cup R$.*

Lemma 3 *(Once adjacent, always adjacent). In the j-th iteration of algorithm* Approx-CMSR*, for $j = 1, 2, \ldots, m$, if two letters a_j and b_j (at least one of them is an isolate of V_j, while the other can be in V_j too or inside an existing substring) are made adjacent into a common substring, then a_j and b_j are maintained adjacent toward the termination of the algorithm. Moreover, in the two original sequences G_1 and G_2, all the letters in between a_j and b_j belong to $\bigcup_{i=1}^{j} U_i$.*

In the sequel, we estimate the size of $U \cup R$ in terms of the size of *OPT*. We do this by attributing all isolates of $U \cup R$ to the letters of *OPT*, through a *local amortized analysis*, and prove that every letter of *OPT* is attributed with at most $\frac{7}{3}$ isolates of $U \cup R$. Note that every letter in $U \cup R$ is an isolate; for the local amortized analysis to be done, we perform the following *re-weighting scheme* to move a fraction of isolatedness of a letter of U to some *related* letters of V, and subsequently to attribute all letters of $U \cup V \cup R$ to the letters of *OPT*. We will prove that every letter of *OPT* is attributed with at most $\frac{7}{3}$ isolates, summing over all the letters of $U \cup V \cup R$ attributed to it.

For the sake of clarity, from now on we refer to every element of $U \cup V \cup R$ as a *letter*, which carries a certain amount of isolatedness of $U \cup R$. Consider U_j. At the beginning, a letter $x \in U_j$ carries 1 isolate of $U \cup R$. If the j-th iteration is a high or a medium priority operation, we change letter x to carry only $\frac{1}{3}$ isolate, while re-distribute the other $\frac{2}{3}$ isolates to two related letters of V_j, $\frac{1}{3}$ each. If the j-th iteration is a low priority operation, we change letter x to carry only $\frac{2}{3}$ isolates, while re-distribute the other $\frac{1}{3}$ isolate to a unique related letter of V_j. The detailed such re-weighting scheme is presented in the last column of Table 1. Two key remarks: every letter of V_j receives 0 or $\frac{1}{3}$ isolate, and if it receives $\frac{1}{3}$, it is also called a $\frac{1}{3}$-isolate of $U \cup R$; every letter of U_j keeping $\frac{1}{3}$ isolate is called a $\frac{1}{3}$-isolate, and grouped into set U^1, while every letter of U_j keeping $\frac{2}{3}$ isolates is called a $\frac{2}{3}$-isolate, and grouped into set U^2. Notation-wise, we use U_j^1 to indicate that

a high or medium priority operation is executed by algorithm Approx-CMSR in the j-th iteration; the corresponding V_j is then denoted as V_j^1. Similarly, U_j^2 is used to indicate that a low priority operation is executed by algorithm Approx-CMSR in the j-th iteration; the corresponding V_j is then denoted as V_j^2. In the following, we will attribute the total isolatedness carried by the letters of $U \cup V \cup R$, which is equal to the size of $U \cup R$, to the letters of *OPT*, through a local amortized analysis, and prove that every letter of *OPT* is attributed with at most $\frac{7}{3}$ isolates of $U \cup R$.

Consider the inverse process of deleting units of *OPT* from G_1 and G_2 to obtain the optimal subsequences 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 inverse process, there are no isolated letters in G_1^* and G_2^*; each letter of $U \cup V \cup R$ thus either is a unit of O_1, or is in some substring of G_1^* and G_2^* but then *singled* out at some point of the inverse process when inserting a unit of *OPT* back into G_1^* and G_2^*, which breaks the substring (or one of its fragments if already being broken) into fragments, one of which is this single letter. In either case, this letter of $U \cup V \cup R$ is said *generated* by the inserting unit of *OPT* and some portion of the isolatedness carried by this letter is *attributed* to the letters in the inserting unit. Since there could be multiple units of *OPT* that are able to generate this letter of $U \cup V \cup R$, we will set up a rule on how to attribute all the carried isolatedness to the letters in these units of *OPT*.

At any time of the inverse process, inserting one unit of *OPT* back to the current sequences G_1 and G_2 can break at most two adjacencies, each by a copy of the inserting unit. Here an adjacency $a \bullet b$ means that $a \bullet b$ is in a common substring of the current sequences G_1 and G_2, and broken means a copy of the inserting unit resides in between a and b in at least one of the two original sequences. It follows that at most four letters of $U \cup V \cup R$, and thus at most four isolates of $U \cup R$, can be generated by the inserting unit. We firstly insert substring units of *OPT*, one by one; all the isolatedness carried by the letters of $U \cup V \cup R$ that are generated by a substring unit of *OPT* is attributed to the letters in the substring unit. Lemma 4 summarizes the fact that every letter of $O_3 \cup O_2$ is attributed with at most 2 isolates of $U \cup R$. The resultant sequences after inserting all substring units of *OPT* are denoted as G_1^0 and G_2^0.

Lemma 4. *Every letter of $O_3 \cup O_2$ is attributed with at most* 2 *isolates of $U \cup R$.*

Proof. Inserting one substring unit of *OPT* back into the current sequences G_1 and G_2 generates at most four letters of $U \cup V \cup R$, each is at most 1-isolate of $U \cup R$. Note that every substring unit contains at least two letters. Therefore, every letter of $O_3 \cup O_2$ can be attributed with at most 2 isolates of $U \cup R$. □

Recall that we use an *isolate/letter copy* to refer to the isolate/letter in one of the two sequences, while reserve *isolate/letter* to refer to both its copies as a whole.

Lemma 5. *In G_1^0 and G_2^0, if $a \bullet b$ is an adjacency for some letter $a \in U \cup V \cup R$, then no letters of any type-*0 *substring can reside in between a and b in any of the original sequences G_1 and G_2.*

Proof. Note that every type-0 substring of G_1 and G_2 is present in G_1^0 and G_2^0. Assume to the contrary that, there is a letter s from a type-0 substring S that resides in between a and b in G_1. From the fact that S is present in G_1^0 and G_2^0, so is s. It follows that $a \bullet b$ cannot be an adjacency in G_1^0 and G_2^0. □

Lemma 6. *In G_1^0 and G_2^0, if $s \bullet v$ is an adjacency for some letters $s \notin U \cup V \cup R$, $v \in V$, then there is at least one isolate copy of O_1 that breaks this adjacency.*

Proof. Since v is an isolate, there must be some letter that resides in between s and v in at least one of G_1 and G_2. Lemma 5 tells that this letter is an isolate, and thus belongs to O_1. This proves the lemma that there is at least one isolate copy of O_1 that breaks this adjacency $s \bullet v$. We choose, arbitrarily, one such isolate copy of O_1 and attribute to it the isolatedness carried by v (see row 1 of Table 2). □

Lemma 7. *In G_1^0 and G_2^0, if $v_1 \bullet v_2$ is an adjacency for some letters $v_1, v_2 \in V$, then there is at least one isolate copy of O_1 that breaks this adjacency.*

Proof. The same proof as for Lemma 6 applies, by using the fact that v_1 is an isolate. Again, we choose arbitrarily one such isolate copy of O_1 and attribute to it the isolatedness carried by v_1 and v_2 (see row 2 of Table 2). □

Lemma 8. *In G_1^0 and G_2^0, if $u \bullet v$ is an adjacency for letters $u \in U_j$ and $v \in V$, for some $j = 1, 2, \ldots, m$, then either there is one isolate copy of $O_1 \cap V_j$ that breaks this adjacency, or there are at least two isolate copies of $O_1 \cap V$ that break this adjacency.*

Proof. Assume first $v \in V_j$. That is, in the j-th iteration of algorithm Approx-CMSR, u is deleted while v is kept. Assume v is made adjacent to a in the j-th iteration. It follows that at the beginning of the j-th iteration of algorithm Approx-CMSR,

Table 2
The scheme for attributing the total amount of isolatedness of the two letters in the adjacency to the breaking isolate copies of O_1, based on Lemmas 6–16.

Adjacency	Lemma	Letter sources	Breaking isolate copies of O_1	Attribution
$s \bullet v$	6	$s \notin U \cup V \cup R$, $v \in V$	$o \in O_1$	$\leqslant \frac{1}{3}$
$v_1 \bullet v_2$	7	$v_1, v_2 \in V$	$o \in O_1$	$\leqslant \frac{2}{3}$
$u \bullet v$	8	$u \in U_j$, $v \in V$	$a \in O_1 \cap V_j$	$\leqslant 1$
			$a_1, a_2 \in O_1 \cap V$	$\leqslant \frac{1}{2}$ each
$r \bullet v$	9	$r \in R$, $v \in V$	$o_1, o_2 \in O_1$	$\leqslant \frac{2}{3}$ each
$s \bullet u$	10	$s \notin U \cup V \cup R$, $u \in U$	$a \in O_1 \cap V$	$\leqslant \frac{2}{3}$
$u_1 \bullet u_2$	11	$u_1, u_2 \in U_j^1$	$a \in O_1 \cap V_j$	$\frac{2}{3}$
	12	$u_1, u_2 \in U_j^2$	$o_1, o_2 \in O_1$	$\frac{2}{3}$ each
	13	$u_1 \in U_i^1, u_2 \in U_j$ $(i < j)$	$a \in O_1 \cap V_i$	$\leqslant 1$
		$u_1 \in U_i, u_2 \in U_j$ $(i < j)$	$o_1, o_2 \in O_1$	$\leqslant \frac{2}{3}$ each
$r \bullet u$	14	$r \in R, u \in U_j^1$	$a \in O_1 \cap V_j$	$\frac{4}{3}$
		$r \in R, u \in U_j$	$a \in O_1 \cap V$, $o \in O_1$	$\frac{2}{3}$ to o
				$\leqslant 1$ to a
$s \bullet r$	15	$s \notin U \cup V \cup R$, $r \in R$	$o_1, o_2 \in O_1$	$\frac{1}{2}$ each
$r_1 \bullet r_2$	16	$r_1, r_2 \in R$	$o_1, o_2, o_3 \in O_1$	$\leqslant \frac{2}{3}$ each
			$a_1, a_2 \in O_1 \cap V$	1 each

u and v do not form a common substring in the two sequences. Since algorithm Approx-CMSR is impossible to execute an operation 1 in the j-th iteration, u is in between v and a in exactly one of the two sequences, or equivalently a has to be in between u and v in exactly one of the two sequences. If $a \in V_j$, the lemma is proved; otherwise, a is in an existing type-1 substring S at the beginning of the j-th iteration, and thus this substring S resides in between u and v in one of the two sequences. Note that a type-1 substring contains at least two isolates of V. This proves there are at least two isolate copies of $O_1 \cap V$ that break adjacency $u \bullet v$.

Secondly, assume $v \in V_i$ for some $i > j$. Assume also that in the j-th iteration of algorithm Approx-CMSR u is deleted to merge a and b. We conclude from Lemma 3 that v does not reside in between a and b in either of the two sequences. Equivalently, at least one of a and b, say a, resides in between u and v in at least one of the two original sequences. If $a \in V_j$, the lemma is proved; otherwise, a is in an existing type-1 substring S at the beginning of the j-th iteration, and thus this substring S resides in between u and v in one of the two sequences. Note that a type-1 substring contains at least two isolates of V. This proves there are at least two isolate copies of $O_1 \cap V$ that break adjacency $u \bullet v$.

Lastly, assume $v \in V_i$ for some $i < j$. Note that at the beginning of the j-th iteration of algorithm Approx-CMSR, u and v do not form a common substring in the two sequences. Assume that in the j-th iteration of algorithm Approx-CMSR, u is deleted to merge a and b. If $\{a, b\} \subseteq V_j$ or $v \notin \{a, b\}$, then we conclude that v does not reside in between a and b in either of the two sequences. Equivalently, at least one of a and b, say a, resides in between u and v in at least one of the two original sequences. If $a \in V_j$, the lemma is proved; otherwise, a is in an existing type-1 substring S at the beginning of the j-th iteration, and thus this substring S resides in between u and v in one of the two sequences. Note that a type-1 substring contains at least two isolates of V. This proves there are at least two isolate copies of $O_1 \cap V$ that break adjacency $u \bullet v$. In the other case, $v \in \{a, b\}$, and assume without loss of generality that $v = b$, which implies that $a \in V_j$. Therefore, algorithm Approx-CMSR executes an operation 6 in the j-th iteration. That is, u is in between a and v in exactly one of the two sequences, or equivalently $a \in V_j$ has to be in between u and v in exactly one of the two sequences. This proves the lemma.

When there is one isolate copy of $O_1 \cap V_j$ that breaks adjacency $u \bullet v$ for $u \in U_j$, we choose one and attribute to it the isolatedness carried by u and v (see row 3 of Table 2); otherwise, we choose arbitrarily two breaking isolate copies of $O_1 \cap V$ and attribute to them the isolatedness carried by u and v, each with no greater than $\frac{1}{2}$ (see row 4 of Table 2). □

Lemma 9. *In G_1^0 and G_2^0, if $r \bullet v$ is an adjacency for some letters $r \in R$ and $v \in V$, then there are at least two isolate copies of O_1 that break this adjacency.*

Proof. Note that at the end of the m-th iteration of algorithm Approx-CMSR, both r and v are in the two sequences. Yet algorithm Approx-CMSR finds no feasible operations to execute. If there is a substring separating r and v in at least one of the two sequences, this separating substring must be a type-1 substring containing at least two isolates of V; hence the lemma is proved. Otherwise, there are at least two isolate copies of R separating r and v in the two sequences; again the lemma is proved. We choose arbitrarily two such breaking isolate copies of O_1 and attribute to them the isolatedness carried by r and v, each with no greater than $\frac{2}{3}$ (see row 5 of Table 2). □

Lemma 10. *In G_1^0 and G_2^0, if $s \bullet u$ is an adjacency for letters $s \notin U \cup V \cup R$ and $u \in U$, then there is one isolate copy of $O_1 \cap V$ that breaks this adjacency.*

Proof. Assume $u \in U_j$ is deleted by algorithm Approx-CMSR in the j-th iteration to merge a and b, for some $j = 1, 2, \ldots, m$. If $s \notin \{a, b\}$, then s does not reside in between a and b in either of the two sequences. Therefore, in at least one of the two original sequences, one of a and b resides in between s and u. By Lemma 5, this breaking isolate copy is in $O_1 \cap V$, and the lemma is proved.

In the other case where $s \in \{a, b\}$, assume without loss of generality that $s = b$. This implies that $a \in V_j$ and algorithm Approx-CMSR executes an operation 6 in the j-th iteration. That is, u is in between a and s in exactly one of the two sequences, or equivalently $a \in V_j$ has to be in between s and u in exactly one of the two sequences. The lemma is proved too. One arbitrarily chosen such breaking isolate copy of $O_1 \cap V$ is attributed with the isolatedness carried by u (see row 6 of Table 2). □

Lemma 11. *In G_1^0 and G_2^0, if $u_1 \bullet u_2$ is an adjacency for letters $u_1, u_2 \in U_j$, for some $j = 1, 2, \ldots, m$ where a high priority operation is executed, then there is one isolate copy of $O_1 \cap V_j$ that breaks this adjacency.*

Proof. Since a high priority operation 2 is executed by algorithm Approx-CMSR in the j-th iteration, we conclude that there is exactly one isolate x of V_j in between u_1 and u_2 in one of the two sequences at the beginning of the j-th iteration, a copy of which is attributed with the isolatedness carried by u_1 and u_2 (row 7 of Table 2). This proves the lemma. □

Lemma 12. *In G_1^0 and G_2^0, if $u_1 \bullet u_2$ is an adjacency for letters $u_1, u_2 \in U_j$, for some $j = 1, 2, \ldots, m$ where a low priority operation is executed, then there are at least two isolate copies of O_1 that break this adjacency.*

Proof. Assume that in the j-th iteration of algorithm Approx-CMSR, u_1 and u_2 are deleted to form a substring denoted as $a \bullet b$. Either an operation 4 or an operation 5 is executed. If this is an operation 4, due to its lowest priority we conclude that at the beginning of the j-th iteration of algorithm Approx-CMSR, u_1 and u_2 are separated by at least two isolates in one of the two sequences; hence the lemma is proved. If this is an operation 5, then each of a and b is in between u_1 and u_2 in exactly one of the two sequences; again the lemma is proved. We choose arbitrarily two such breaking isolate copies of O_1 and attribute to them the isolatedness carried by u_1 and u_2, each with $\frac{2}{3}$ (row 8 of Table 2). □

Lemma 13. *In G_1^0 and G_2^0, if $u_1 \bullet u_2$ is an adjacency for letters $u_1 \in U_i$ and $u_2 \in U_j$, for some $i, j = 1, 2, \ldots, m$ with $i < j$, then either there is one isolate copy of $O_1 \cap V_i$ that breaks this adjacency, or there are at least two isolate copies of $O_1 \cap V$ that break this adjacency. Furthermore, if there is only one isolate copy of O_1 that breaks this adjacency, then algorithm* Approx-CMSR *must execute a high or medium priority operation in the i-th iteration (i.e., u_1 is a $\frac{1}{3}$-isolate).*

Proof. Assume that u_1 is deleted by algorithm Approx-CMSR in the i-th iteration to merge a and b. It follows from $i < j$ that u_2 does not reside in between a and b in either of the two sequences, or equivalently at least one of a and b, say a resides in between u_1 and u_2 in at least one of the two original sequences. If $a \in V_i$, the lemma is proved; otherwise, a is in an existing type-1 substring S at the beginning of the i-th iteration, and thus this substring S resides in between u_1 and u_2 in one of the two sequences. Note that a type-1 substring contains at least two isolates of V. This proves there are at least two isolate copies of $O_1 \cap V$ that break adjacency $u_1 \bullet u_2$.

Furthermore, the above proof says that when there is exactly one isolate copy of O_1, which is a copy of a, breaking adjacency $u_1 \bullet u_2$, then algorithm Approx-CMSR must execute a high or medium priority operation to delete u_1, since otherwise it would have to execute a medium priority operation 3 to delete isolate a and to merge u_1 and u_2. This proves the lemma.

Rows 9 and 10 in Table 2 are associated with the scenarios covered by this lemma. When there is only one isolate copy of O_1 that breaks adjacency $u_1 \bullet u_2$, a high or medium priority operation is executed in the i-th iteration and thus u_1 is a $\frac{1}{3}$-isolate; consequently this breaking isolate copy of O_1 is attributed with at most 1 isolate of $U \cup R$ (row 9). When there are more than one isolate copy of O_1 that break adjacency $u_1 \bullet u_2$, we select two arbitrarily and attribute to them the isolatedness carried by u_1 and u_2, each with at most $\frac{2}{3}$ (row 10). □

Lemma 14. *In G_1^0 and G_2^0, if $r \bullet u$ is an adjacency for letters $r \in R$ and $u \in U_j$, for some $j = 1, 2, \ldots, m$, then either there is one isolate copy of $O_1 \cap V_j$ that breaks this adjacency, or there are at least two isolate copies of $O_1 \cap V$ that break this adjacency. Furthermore, if there is only one isolate copy of O_1 that breaks this adjacency, then algorithm* Approx-CMSR *executes a high or medium priority operation in the j-th iteration (i.e., $u \in U^1$).*

Proof. Assume that u is deleted by algorithm Approx-CMSR in the j-th iteration to merge a and b. It follows that r does not reside in between a and b in either of the two sequences, or equivalently at least one of a and b, say a, resides in between r and u in at least one of the two original sequences. If $a \in V_j$, the lemma is proved; otherwise, a is in an existing type-1 substring S at the beginning of the j-th iteration, and thus this substring S resides in between r and u in one of the two sequences. Note that a type-1 substring contains at least two isolates of V. This proves there are at least two isolate copies of $O_1 \cap V$ that break adjacency $r \bullet u$.

Furthermore, the above proof says that when there is exactly one isolate copy of O_1, which is a copy of a, breaking adjacency $r \bullet u$, then algorithm Approx-CMSR must execute a high or medium priority operation to delete u, since otherwise it would have to execute a medium priority operation 3 to delete isolate a and to merge r and u. This proves the lemma.

Rows 11 and 12 in Table 2 are associated with the scenarios covered by this lemma. When there is only one isolate copy of O_1 that breaks adjacency $r \bullet u$, a high or medium priority operation is executed in the i-th iteration and thus u is a $\frac{1}{3}$-isolate; consequently this breaking isolate copy of O_1 is attributed with $\frac{4}{3}$ isolates of $U \cup R$ (row 11). When there are more than one isolate copy of O_1 that break adjacency $r \bullet u$, at least one of them belongs to V. We select one isolate copy a of $O_1 \cap V$ and the other o arbitrarily and attribute to them the isolatedness carried by r and u; we always attribute $\frac{2}{3}$ to o and the rest to a, which is either 1 or $\frac{2}{3}$ (row 12). □

Lemma 15. *In G_1^0 and G_2^0, if $s \bullet r$ is an adjacency for some letters $s \notin U \cup V \cup R$, $r \in R$, then there are at least two isolate copies of O_1 that break this adjacency.*

Proof. If there is no isolate of V that resides in between s and r in either of the two original sequences, then due to the fact that at the end of the m-th iteration, both s and r are present in the two sequences while algorithm Approx-CMSR finds no feasible operation to merge them, we conclude there must be at least two isolate copies of $O_1 \cap R$ that break this adjacency. In the other case, there is a type-1 substring that resides in between s and r in one of the two sequences at the end of the m-th iteration. Note that a type-1 substring contains at least two letters of V. Therefore, there must be at least two isolate copies of $O_1 \cap V$ that break this adjacency. This proves the lemma. We choose arbitrarily two of the breaking isolate copies and attribute to them the isolatedness carried by r, each with $\frac{1}{2}$ (row 13 of Table 2). □

Lemma 16. *In G_1^0 and G_2^0, if $r_1 \bullet r_2$ is an adjacency for some letters $r_1, r_2 \in R$, then either there are at least three isolate copies of O_1 that break this adjacency, or there are at least two isolate copies of $O_1 \cap V$ that break this adjacency.*

Proof. If there is no isolate of V that resides in between r_1 and r_2 in either of the two original sequences, then due to the fact that at the end of the m-th iteration, both r_1 and r_2 are present in the two sequences while algorithm Approx-CMSR finds no feasible operation to merge them, we conclude there must be at least three isolate copies of $O_1 \cap R$ that break this adjacency. In the other case, there is a type-1 substring that resides in between r_1 and r_2 in one of the two sequences at the end of the m-th iteration. Note that a type-1 substring contains at least two letters of V. Therefore, there must be at least two isolate copies of $O_1 \cap V$ that break this adjacency. (In fact, we can prove further that there are at least two isolate copies of $O_1 \cap V_j$, for some $j = 1, 2, \ldots, m$, that break this adjacency.) This proves the lemma.

The last two rows in Table 2, that is rows 14 and 15, are associated with the scenarios covered by this lemma. When there are three or more isolate copies of O_1 that break adjacency $r_1 \bullet r_2$, arbitrarily pick three and each of them is attributed with $\frac{2}{3}$ isolates of $U \cup R$ (row 14). Otherwise, each of the two breaking isolate copies of $O_1 \cap V$ is attributed with 1 isolate of $U \cup R$ (row 15). □

Note that after all units of O_1 are inserted into G_1^0 and G_2^0 using their positions in the two original sequences, all those adjacencies in G_1^0 and G_2^0 involving a letter of $U \cup V \cup R$ are broken. Lemmas 6–16 identify all possible scenarios of such adjacencies, to determine the number of isolate copies of O_1, and their memberships of U, V or R whenever possible, that should break each kind of adjacency. Subsequently, the total isolatedness carried by the two letters in the adjacency can be attributed to the (selected) breaking isolate copies of O_1. Table 2 presents the attribution scheme and Lemma 17 summarizes some of the consequences.

Lemma 17. *Using the isolatedness attribution scheme presented in* Table 2*, we have*

i) *every isolate copy of $O_1 \cap (U \cup R)$ is attributed with at most $\frac{2}{3}$ isolates of $U \cup R$;*
ii) *every isolate copy of $O_1 \cap V_j^2$ is attributed with at most* 1 *isolate of $U \cup R$;*
iii) *every isolate copy of $O_1 \cap V_j^1$, on average, is attributed with at most* 1 *isolate of $U \cup R$.*

Proof. From Table 2, one can see that an isolate copy attributed with more than $\frac{2}{3}$ isolates occurs in rows 3, 9, 11, 12 and 15, each of which belongs to $O_1 \cap V$. Therefore, every isolate copy of $O_1 \cap (U \cup R)$ is attributed with at most $\frac{2}{3}$ isolates. This proves the first part i).

Note that there is only one scenario where an isolate copy $a \in V_j$ may be attributed with more than 1 isolate, in row 11 of Table 2, or Lemma 14, where algorithm Approx-CMSR executes a high or medium priority operation in the j-th iteration. Therefore, every isolate copy of $O_1 \cap V_j^2$ is attributed with at most 1 isolate. This proves the second part ii).

To prove the last part iii), we examine more carefully the scenario in Lemma 14, where the isolate copy $a \in O_1 \cap V_j$ is attributed with $\frac{4}{3}$ isolates of $U \cup R$. In this scenario, $r \in R$, $u \in U_j^1$ for some $j = 1, 2, \ldots, m$, $r \bullet u$ is the adjacency in G_1^0 and G_2^0, and a is the only isolate copy breaking this adjacency. Additionally, in the j-th iteration algorithm Approx-CMSR

deletes u to merge the two isolates a and b. For ease of presentation we further assume that $r \bullet u$ occurs in the original sequence G_1. It follows that $r \bullet a \bullet u$ occurs in the original sequence G_2. Note that in an operation 1 or an operation 2, the deleted letter u (or in its reversed and negated form) should not be adjacent to any letter of R, in any of the two original sequences G_1 and G_2. We conclude that in the j-th iteration of algorithm Approx-CMSR, a medium operation 3 is executed. Therefore, $U_j^1 = \{u\}$ and $V_j \supseteq \{a, b\}$ (row 3 in Table 1). We consider two cases.

Case 1: $b \in O_1$. That is, in this case, b is also deleted in the optimal solution.

If the copy of b in the original sequence G_2, denoted as b'', does not break any adjacency in G_1^0 and G_2^0, it is attributed with 0 isolate; if b'' breaks an adjacency denoted as $s \bullet t$ in G_1^0 and G_2^0, then due to the fact that u is in G_1^0 and G_2^0 we know that exactly one of s and t, say s, resides in between u, inclusive, and b'' in the original sequence G_2. It follows from Lemma 3 that $s \in \bigcup_{i=1}^{j} U_i$.

Case 1.1: $s = u$. In this situation, letter s carries no more extra isolatedness in the adjacency $s \bullet t$, as its isolatedness has been counted in the adjacency $r \bullet u$ (*i.e.*, $r \bullet s$). If $t \notin R$, then it is at most a $\frac{2}{3}$-isolate, and thus b'' is attributed with at most $\frac{2}{3}$ isolates; If $t \in R$, b'' is attributed with at most 1 isolate, and by Lemma 14 it is attributed with exactly 1 isolate only if it is the only isolate copy of O_1 breaking adjacency $u \bullet t$ (row 9 in Table 2), or otherwise it is attributed with at most $\frac{1}{2}$ isolate (row 10 in Table 2, to combine with the fact that s is a $\frac{1}{3}$-isolate).

Case 1.2: $s \neq u$. In this situation, letter $s \in \bigcup_{i=1}^{j-1} U_i$ and thus it is at most a $\frac{2}{3}$-isolate. If $t \notin U \cup V \cup R$, then b'' is attributed with at most $\frac{2}{3}$ isolates. If $t \in U_k$ for some $k = 1, 2, \ldots, m$, then $k \neq j$. If it happens that $s \in U_k$ too, Lemma 11 says that b'' is not attributed with any isolate since $b'' \notin V_k$ (note that row 7 in Table 2 applies to b'' only if b'' were in V_k); if $s \in U_i$ for some $i \neq k$, for the same reason that $b'' \notin V_i \cup V_k$, Lemma 13 says that b'' is attributed with at most $\frac{2}{3}$ isolates (*i.e.*, row 9 in Table 2 does not, but row 10 applies to b''). If $t \in V$, Lemma 8 says that b'' is attributed with at most $\frac{1}{2}$ isolate (*i.e.*, row 3 in Table 2 does not, but row 4 applies to b''); The last case of $t \in R$ is discussed in Lemma 14. Assume without loss of generality that $s \in U_i$ for some $i < j$. Again, since $b'' \notin V_i$, the lemma says that b'' is not the only isolate copy breaking adjacency $s \bullet t$; but a copy of a letter of V_k for some $k \leqslant i$ breaks adjacency $s \bullet t$ as well. Consequently, we can pick b'' as the breaking isolate copy o as in row 12 in Table 2, which is subsequently attributed with $\frac{2}{3}$ isolates.

In summary, the above two subcases tell that b'' is attributed with at most $\frac{2}{3}$ isolates of $U \cup R$, unless otherwise it is the only breaking isolate copy for the adjacency $u \bullet t$ for some $t \in R$, where it is attributed with 1 isolate. The configuration of this latter scenario is that, $r \bullet u \bullet t$ occurs in the original sequence G_1, and $r \bullet a \bullet u \bullet b \bullet t$ occurs in the original sequence G_2. Recall that $r \in R$ too.

Let a' and b' denote the copy of letter a and the copy of letter b in the original sequence G_1. If they do not break a common adjacency in G_1^0 and G_2^0, then each of them breaks an adjacency of which exactly one letter resides in between a and b in the original sequence G_1. Assume that a' breaks the adjacency $w \bullet z$, and further that z resides in between a and b in the original sequence G_1. It follows from Lemma 3 that $z \in \bigcup_{i=1}^{j-1} U_i$. Subsequently, the above discussion of Case 1.2 applies to a', by replacing the triple (b'', s, t) with (a', z, w); that is, a' is attributed with at most $\frac{2}{3}$ isolates of $U \cup R$. The same discussion applies to b' as well, and thus b' is attributed with at most $\frac{2}{3}$ isolates of $U \cup R$ too. Therefore, in total, all four copies of a and b are attributed with at most $\frac{4}{3} + 1 + \frac{2}{3} + \frac{2}{3} = \frac{11}{3}$ isolates.

Assume next a' and b' break a common adjacency denoted as $w \bullet z$, that is, both a' and b' reside in between w and z in the original sequence G_1. When b'' is attributed with at most $\frac{2}{3}$ isolates, or one of w and z is not in R, all four copies of a and b are attributed with at most $\max\{\frac{4}{3} + \frac{2}{3} + 2, \frac{4}{3} + 1 + \frac{5}{3}\} = 4$ isolates. In the other case, b'' is attributed with exactly 1 isolate and $w, z \in R$. From the above two subcases Case 1.1 and Case 1.2, b'' is the only isolate copy breaking adjacency $u \bullet t$, *i.e.*, $r \bullet u \bullet t$ occurs in the original sequence G_1, and $r \bullet a \bullet u \bullet b \bullet t$ occurs in the original sequence G_2. We conclude that there must be another isolate copy of O_1 that also breaks adjacency $w \bullet z$. For contradiction assuming a' and b' are the only isolate copies breaking $w \bullet z$, *i.e.*, $w \bullet a \bullet b \bullet z$ occurs in the original sequence G_1, then in the j-th iteration algorithm Approx-CMSR would have to execute a high priority operation 2 to delete a and b and to form substrings $r \bullet u \bullet t$ and $w \bullet z$, instead of a medium priority operation 3 to delete u and to form a substring $a \bullet b$. By Lemma 16 (the second last row in Table 2), each of a' and b' is attributed with at most $\frac{2}{3}$ isolates of $U \cup R$. Subsequently, all four copies of a and b are attributed with at most $\frac{4}{3} + 1 + \frac{2}{3} + \frac{2}{3} = \frac{11}{3}$ isolates. This proves the part iii) in Case 1.

Case 2: $b \notin O_1$. That is, b is in G_1^0 and G_2^0.

In this case, we only need to discuss the copy of a in the original sequence G_1, denoted as a'. Due to the presence of b in G_1^0 and G_2^0, a' either does not break any adjacency and thus is attributed with 0 isolate, or breaks an adjacency of which exactly one letter resides in between a and b, inclusive, in the original sequence G_1. Assume that a' breaks the adjacency $w \bullet z$, and further that z resides in between a and b, inclusive, in the original sequence G_1. Since a median priority operation is executed in the j-th iteration to delete u, we have $z \neq u$; it follows from Lemma 3 that $z \in \bigcup_{i=1}^{j-1} U_i$. Subsequently, the above discussion of Case 1.2 applies to a', by replacing the triple (b'', s, t) with (a', z, w); that is, a' is attributed with at most $\frac{2}{3}$ isolates of $U \cup R$. Therefore, the two copies of a together are attributed with at most $\frac{4}{3} + \frac{2}{3} = 2$ isolates of $U \cup R$. This proves the part iii) in Case 2. □

Theorem 1. *Algorithm* Approx-CMSR *is a* $\frac{7}{3}$*-approximation algorithm for the CMSR problem.*

Proof. Approx-CMSR runs in polynomial time in n, where n is the number of letters in the input sequences, since in each iteration of the second step the searching-deleting-retaining operation can be done in $O(n^2)$ time and there are $O(n)$ iterations.

From the part i) of Lemma 17, that every isolate copy of $O_1 \cap (U \cup R)$ is attributed with at most $\frac{2}{3}$ isolates, every letter of $O_1 \cap (U \cup R)$ is attributed with at most $\frac{2}{3} \times 2 + 1 = \frac{7}{3}$ isolates, where 1 comes from the fact that the isolatedness carried by the letter is attributed to itself. From the part ii), that every isolate copy of $O_1 \cap V_j^2$ is attributed with at most 1 isolate, every isolate of $O_1 \cap V_j^2$ is attributed with at most $1 \times 2 + \frac{1}{3} = \frac{7}{3}$ isolates, where $\frac{1}{3}$ comes from the fact that the isolatedness carried by the letter is attributed to itself. From the part iii) of Lemma 17, for every isolate $v \in O_1 \cap V_j^1$, its two copies together are attributed with at most 2 isolates, and thus it is attributed with at most $2 + \frac{1}{3} = \frac{7}{3}$ isolates, where again $\frac{1}{3}$ comes from the fact that the isolatedness carried by the letter is attributed to itself. These and Lemma 4 imply that $|U \cup R| \leqslant \frac{7}{3}|OPT|$. Note from Lemma 2 that $U \cup R$ is the set of isolates deleted by algorithm Approx-CMSR. Therefore, it is a $\frac{7}{3}$-approximation algorithm. □

4. Conclusions

In this paper, we presented a $\frac{7}{3}$-approximation algorithm Approx-CMSR for the CMSR problem. The key design technique is a prioritized local greedy scheme to retain a number of isolates while deleting one or two isolates. The three levels of priorities set for the six operations are crucial in the performance analysis, which is done through a novel local amortized analysis together with a re-weighting scheme.

Briefly, the analysis is done by examining the isolates which are kept in the optimal solution but deleted by algorithm Approx-CMSR, and to attribute them to the letters deleted in the optimal solution. By local amortization, an isolate is attributed to the letters that break the adjacency the isolate is involved. Using the sequential order of the letters, from the algorithm description, one letter could be attributed with as many as 4 isolates. The re-weighting scheme is introduced to move a fraction of isolatedness carried by a deleted letter to the letters kept by the algorithm. This scheme, which is interesting by itself, makes the attribution process much easier to present, to guarantee that one letter is attributed with at most $\frac{7}{3}$ isolates. It is yet to be seen whether these techniques, the prioritized operations, the local amortized analysis, and the re-weighting scheme, can be further developed to incorporate more complex greedy decisions, to design a better approximation algorithm.

The following instance shows that the performance ratio $\frac{7}{3}$ is tight, even for the unsigned version of the CMSR problem. In this instance,

$$G_1 = \langle a, b, c, x_1, x_2, x_3, d, e, f, g, y_1, y_2, y_3, h, i, j, k, z_1, z_2, z_3, \ell \rangle, \quad \text{and}$$
$$G_2 = \langle k, \ell, i, x_1, y_1, z_1, j, c, d, a, x_2, y_2, z_2, b, g, h, e, x_3, y_3, z_3, f \rangle.$$

G_1 and G_2 have no type-0 common substrings, neither does deleting one or two letters from them lead to a novel type-1 common substring. Therefore, Approx-CMSR deletes all the letters. On the other hand, deleting 9 letters, $x_1, x_2, x_3, y_1, y_2, y_3, z_1, z_2, z_3$, leads to

$$G_1^* = \langle a, b, c, d, e, f, g, h, i, j, k, \ell \rangle, \quad \text{and}$$
$$G_2^* = \langle k, \ell, i, j, c, d, a, b, g, h, e, f \rangle,$$

which can be partitioned into six common length-2 substrings: $a \bullet b$, $c \bullet d$, $e \bullet f$, $g \bullet h$, $i \bullet j$, and $k \bullet \ell$. In fact, one can verify that this is an optimal solution to the instance. It follows that the performance ratio of Approx-CMSR on this instance is $\frac{21}{9} = \frac{7}{3}$. Based on this 21-letter instance, one can construct an infinite series of instances on which the performance ratio of Approx-CMSR is $\frac{7}{3}$.

Acknowledgments

This research is partially supported by NSERC. The authors would like to thank Dr. Binhai Zhu, and five reviewers from the Fifth International Frontiers of Algorithmics Workshop (FAW 2011) and the Seventh International Conference on Algorithmic Aspects of Information and Management (AAIM 2011), for their insightful comments on the extended abstract of this work, which appears in the joint conference proceedings, LNCS 6681, pp. 46–57. The authors are also grateful to the two reviewers for this journal version, for their careful validating and their critical comments, particularly on the proof of Lemma 17, which improve greatly the presentation.

References

[1] R. Bar-Yehuda, M.M. Halldórsson, J.S. Naor, H. Shachnai, I. Shapira, Scheduling split intervals, SIAM J. Comput. 36 (2006) 1–15.
[2] L. Bulteau, G. Fertin, I. Rusu, Maximal strip recovery problem with gaps: hardness and approximation algorithms, in: Proceedings of the 20th Annual International Symposium on Algorithms and Computation (ISAAC'09), in: Lecture Notes in Comput. Sci., vol. 5878, 2009, pp. 710–719.

[3] Z. Chen, B. Fu, M. Jiang, B. Zhu, On recovering synthetic blocks from comparative maps, J. Comb. Optim. 18 (2009) 307–318.
[4] V. Choi, C. Zheng, Q. Zhu, 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), 2007, pp. 277–288.
[5] H. Jiang, Z. Li, G. Lin, L. Wang, B. Zhu, Exact and approximation algorithms for the complementary maximal strip recovery problem, J. Comb. Optim. (2010), doi:10.1007/s10878-010-9366-y, in press.
[6] M. Jiang, Inapproximability of maximal strip recovery, in: Proceedings of the 20th Annual International Symposium on Algorithms and Computation (ISAAC'09), in: Lecture Notes in Comput. Sci., vol. 5878, 2009, pp. 616–625.
[7] M. Jiang, Inapproximability of maximal strip recovery, II, in: Proceedings of the 4th Annual Frontiers of Algorithmics Workshop (FAW'10), in: Lecture Notes in Comput. Sci., vol. 6213, 2010, pp. 53–64.
[8] L. Wang, B. Zhu, On the tractability of maximal strip recovery, J. Comput. Biol. 17 (2010) 907–914;
L. Wang, B. Zhu, J. Comput. Biol. 18 (2011) 129 (Correction).
[9] C. Zheng, Q. Zhu, D. Sankoff, Removing noise and ambiguities from comparative maps in rearrangement analysis, IEEE/ACM Trans. Comput. Biology Bioinform. 4 (2007) 515–522.