A Linear Kernel for the Complementary Maximal Strip Recovery Problem

Haitao Jiang[1,2] and Binhai Zhu[3]

[1] School of Computer Science and Technology, Shandong University, Jinan, Shandong 250100, China. Email: htjiang@mail.sdu.edu.cn
[2] School of Mathematics and System Science, Shandong University, Jinan, Shandong 250100, China.
[3] Department of Computer Science, Montana State University, Bozeman, MT 59717-3880, USA. Email: bhz@cs.montana.edu

Abstract. In this paper, we compute the first linear kernel for the complementary problem of Maximal Strip Recovery (CMSR) — a well-known NP-complete problem in computational genomics. Let k be the parameter which represents the size of the solution. The core of the technique is to first obtain a tight $18k$ bound on the parameterized solution search space, which is done through a mixed global rules and local rules, and via an inverse amortized analysis. Then we apply additional data-reduction rules to obtain a tight $78k$ kernel for the problem. Combined with the known algorithm using bounded degree search, we obtain the best FPT algorithm for CMSR to this date, running in $O(2.36^k k^2 + n^2)$ time.

1 Introduction

The rapid development of the parameterized complexity theory greatly enhances our understanding beyond NP-completeness and the traditional computational complexity theory [6, 22, 13]. For many theoretically intractable applications, FPT (fixed-parameter tractable) algorithms can be very effective [7, 11, 21].

In the parameterized complexity theory, kernelization is a very useful tool [9, 14]. Loosely, kernelization means the reduction of the problem instance size to a function of k (k is the parameter throughout this paper). In reality, small (especially small linear) kernel can make it feasible to use some traditional method like branch-and-bound or ILP, so it is always meaningful. On the other hand, there are various problems which do not admit small (or even polynomial) kernels unless the polynomial hierarchy collapses to its third level [1, 8, 10, 12].

In the Complementary Maximal Strip Recovery (CMSR) problem, we need to delete at most k letters from the two input sequences (signed permutations) such that the remaining letters all form into strips (or maximal common substrings of length at least two, some could be in negated and reversed form). To this date, there are two bounded search tree algorithms running in $O^*(3^k)$ [17] and $O^*(2.36^k)$ [3] respectively for CMSR, but no (linear or even polynomial) kernel is known. Part of the reason that a (linear) kernel is elusive for the CMSR is that

the only known local rule (see Lemma 1, i.e., 'long' maximal common substrings can be kept as strips) is not enough to establish any polynomial kernel.

In this paper, we obtain a linear $78k$ kernel for CMSR. The core of our idea is to first bound the *parameterized* solution search space (i.e., the set of letters, whose size is a function of k, from which an optimal solution can be obtained). By applying a set of global rules (together with the local rule induced by Lemma 1), we show that this space is of size at most $18k$. On top of this we can build successfully the linear kernel of size $78k$ for CMSR.

This paper is organized as follows. In Section 2, we define the MSR and CMSR problems and the corresponding concepts for FPT formally. In Section 3, we derive the $78k$ kernel bound for CMSR. In Section 4, we close the paper with several open problems.

2 Preliminaries

MSR and CMSR Maximal Strip Recovery (MSR) was a problem originally proposed by the David Sankoff group to eliminate noise and ambiguities in genomic maps [5, 24]. In comparative genomics, a genomic map (interchangeably, a sequence) is represented by a sequence of distinct gene markers (interchangeably, letters). A gene marker can appear in two different genomic maps, in either positive or negative form. A *strip* (syntenic block) is a sequence of distinct markers that appears as subsequences in two maps, either directly or in reversed and negated form. Given two genomic maps G_1 and G_2, the problem *Maximal Strip Recovery* (MSR) [5, 24] is to find two subsequences of d strips (each of length at least two), denoted as $G_i^\star$, for $i = 1, 2$, and find two signed permutations π_i of $\langle 1, \ldots, d\rangle$, such that each sequence $G_i^\star = S_{\pi_i(1)} \ldots S_{\pi_i(d)}$ (here S_{-j} denotes the reversed and negated sequence of S_j) is a subsequence of G_i, and the total length of the strips S_j is maximized. Intuitively, those gene markers not included in $G_1^\star$ and $G_2^\star$ are noise and ambiguities. The complementary problem of deleting the minimum number of noise and ambiguous markers to have a feasible solution (i.e., every remaining marker must be in some strip) is exactly the *complement of MSR*, which will be abbreviated as CMSR.

We refer to Fig. 1 for an example. In this example, each integer represents a marker.

Not surprisingly, in [23], both MSR and CMSR were shown to be NP-complete. Most recently, MSR was shown to be APX-hard [2, 15] and CMSR was also shown to be APX-hard [16]. For positive results, in [5, 24], some heuristic approaches based on MIS and Max Clique were proposed. In [4], a factor-4 polynomial-time approximation algorithm was proposed for MSR. In [17], a factor-3 polynomial-time approximation algorithm was proposed for CMSR and an $O^*(3^k)$ FPT algorithm was proposed for CMSR (the latter improves and corrects an FPT bound in [23]). Recently, the approximation factor for CMSR was improved to 2.33 [20] and the corresponding FPT algorithmic bound was improved to $O^*(2.36^k n^2)$ [3]. In this paper, we will focus only on the complement of MSR, or the CMSR problem.

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

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

FPT and Kernel We now present some definitions regarding FPT algorithms. Basically, a fixed-parameter tractable (FPT) algorithm for a *decision* problem Π with parameter k is an algorithm which solves the problem in $O(f(k)n^c) = O^*(f(k))$ time, where f is any function only on k, n is the input size and c is some fixed constant not related to k. FPT also stands for the set of problems which admit such an algorithm.

A useful technique in parameterized algorithmics is to provide polynomial time executable data-reduction rules that lead to a *problem kernel*. A data-reduction rule replaces (I,k) by an instance (I',k') in polynomial time such that: (1) $|I'| \leq |I|$, $k' \leq k$, (2) (I,k) is a Yes-instance if and only if (I',k') is a Yes-instance, and (3) $|I'| \leq g(k)$ for some function g. $|I'|$ is called the *size* of the kernel for the problem instance (I, k). A set of polynomial-time data-reduction rules for a problem are applied to an instance of the problem to achieve a *reduced* instance termed the *kernel*. A parameterized problem is FPT if and only if there is a polynomial time algorithm applying data-reduction rules that reduce any instance of the problem to a kernelized instance of size $g(k)$. More about parameterized complexity can be found in the monographs [7, 11, 21].

3 A Linear Kernel for CMSR

Our idea for constructing the linear $78k$ kernel for CMSR is based on first identifying the *parameterized solution search space* for CMSR. Formally, a *parameterized solution search space* for the CMSR problem is a subset S of the markers in G_1, G_2 such that we only need to delete k markers in S to obtain some optimal sequences $G_1^\star$ and $G_2^\star$; moreover, $|S| \leq g(k)$ for some function g. Once an S (of size $18k$) is obtained, it is relatively easy to obtain the linear kernel.

3.1 Bounding the Solution Search Space for CMSR

We first need to do some preprocessing. Before any marker is deleted, we can identify all maximal common substrings of length at least one (possibly in negated and reversed form, which will also be called maximal common substrings, or *block* for convenience) of G_1 and G_2. We also call a length-1 maximal common substring (which is a letter) an *isolated* letter or *isolate*. Two substrings are called *neighbors* if there is no other string in between them. The following lemma is proved in [17], and for completeness we include the proof here.

Lemma 1. *[17] Before any marker is deleted, if a length-4 maximal common substring $xyzw$ or $-w-z-y-x$ appears in both G_1 and G_2 (or, if $xyzw$ appears in G_1 and $-w-z-y-x$ appears in G_2, and vice versa), then there is an optimal solution for MSR which has $xyzw$ or $-w-z-y-x$ as a strip.*

Proof. WLOG, we only consider the case when $xyzw$ appears in G_1 and $-w-z-y-x$ appears in G_2. The cases when $xyzw$ ($-w-z-y-x$) appears in both G_1 and G_2 are similar.

Let the length-6 substring in G_1 containing $xyzw$ be $p_1(x)xyzws_1(w)$ and let the length-6 substring in G_2 containing $-w-z-y-x$ be $p_2(w)-w-z-y-xs_2(x)$. Here $p_i(x), s_i(x)$ means the predecessor and successor of x in G_i. When deleting $xyzw$ from G_1 and $-w-z-y-x$ from G_2, at most two new strips can be obtained which could contain $\{p_1(x), s_1(w), p_2(w), s_2(x)\}$ (with a total size of 4). Clearly, retaining $xyzw$ and $-w-z-y-x$ as a strip can give us a solution at least as good as any optimal solution. Hence, the lemma is proven. □

An example for the above lemma is as follows: $G_1 = cdaxyzwbef$ and $G_2 = e-w-z-y-xfcd-b-a$. $xyzw$ appears in G_1, $-w-z-y-x$ appears in G_2. So we have one optimal solution $G_1^\star = cdxyzw$ and $G_2^\star = -w-z-y-xcd$. On the other hand, the optimal solution is not unique as we can select $G_1^+ = cdabef$ and $G_2^+ = efcd-b-a$.

The above lemma holds for maximal common substrings of length greater than 4. Now let us come back to our journey of obtaining a linear kernel for CMSR. Lemma 1 certainly provides a useful local rule to reduce the search space for solving CMSR. The difficulty now is how to handle length-2 and length-3 blocks. For example, let Q be a length-3 block and all P_i's have length 2, then in

$G_1 = xP_1QP_2y \cdot a_1b_1 \cdot a_2b_2 \cdot a_3P_3b_3 \cdot a_4P_4b_4 \cdot -w-z$

$G_2 = zP_3QP_4w \cdot a_4b_4 \cdot a_3b_3 \cdot a_2P_2b_2 \cdot a_1P_1b_1 \cdot -y-x$

the optimal solution in fact deletes Q, P_1, P_2, P_3, P_4. (Dot symbols are used for connection purpose.) Notice that Q has length-3 and has no isolated neighbor at all, yet it has to be deleted for an optimal solution! One could construct another counter-intuitive example where in a continuous (sequence of) length-2/3 blocks, only a part (i.e., not all) of them are deleted. So besides Lemma 1, it is in fact hard to apply any more local rules (with the ones we proposed early on, eventually counter-examples are found for each of them).

It turns out that we have to use a set of global rules together with a general graph method, which is described below in the algorithm.

Let Σ be the alphabet for the input maps G_1 and G_2. The kernelization procedure (for identifying S) is as follows.

1. Without deleting any gene marker in G_1 and G_2, identify a set of maximal common substrings (possibly in reversed and negated form) of length at least 4, of length-3, of length-2 (which are called blocks)and of length-1 (which are called isolates). Then identify all maximal continuous blocks, which is composed of blocks but isolates, in G_1 and G_2. We call them *super-blocks* henceforth, and denote them as $V_1 \in G_1$ and $V_2 \in G_2$. Apparently, *super-blocks* and sequences of isolates appear alternatively in G_1 and G_2 respectively. Then, we can construct a simple bipartite graph $G = (V_1, V_2, E)$, where each vertex in V_1 or V_2 corresponds a super-block of G_1 or G_2, and there is an edge $(v_1, v_2) \in E$ between two super-blocks $v_1 \in V_1, v_2 \in V_2$ iff they share a common length-2 or length-3 block. (See Fig. 2.)

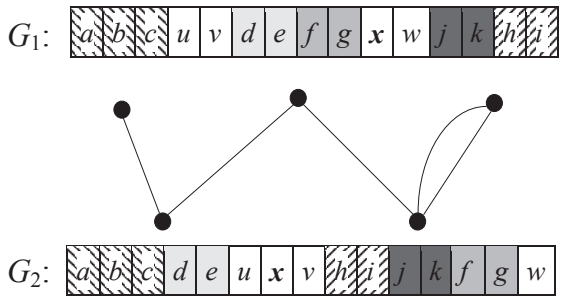

Fig. 2. Blocks, super-blocks, isolates and the bipartite graph. There are five blocks: abc, de, hi, jk, fg**; four isolates:** u, x, v, w**; three super-blocks in** G_1**:** abc, $defg$, $jkhi$ **and two super-blocks in** G_2**:** $abcde$, $hijkfg$**.**

2. *Rule*(2.1) Firstly, for each block of length at least 4, change it to a new letter in Σ_1 (and delete the corresponding old letters in it from Σ whenever such a new letter in Σ_1 is created), with $\Sigma_1 \cap \Sigma = \emptyset$.
   Rule(2.2) Secondly, for any pair of super-blocks $s_1 \in V_1, s_2 \in V_2$ which contain at least two pairs of common length-2 or length-3 blocks, identify the leftmost and rightmost such common blocks in s_1 (e.g., P_i, P_j) and in s_2 (e.g., P_l, P_r, with $P_i = P_l, P_j = P_r$ or $P_i = P_r, P_j = P_l$, some possibly in reversed and negated form). Change each block between and inclusive of P_i, P_j (resp. P_l, P_r) in s_1 (resp. s_2) into a new letter in Σ_1. As shown in Fig. 3, the blocks gh and mn contribute to multiple edges between the two super-blocks $mngh$ and $gh12mnij$ in G.
   Rule(2.3) Thirdly, for any super-block (in V_1 or V_2) containing at least two length-3 blocks, identify the leftmost and rightmost length-3 blocks, say

P_s, P_t. Change each block between and inclusive of P_s, P_t into a new letter in Σ_1, such as the blocks rst and opq in Fig. 3.
Rule(2.4) Then, construct the simple bipartite graph $G = (V_1, V_2, E)$, where there is an edge $(v_1, v_2) \in E$ between two super-blocks $v_1 \in V_1, v_2 \in V_2$ iff they share a common length-2 or length-3 block not yet put in Σ_1. For any cycle in G, identify the length-2 or length-3 blocks involved in the cycle and change each of them into a new letter in Σ_1. As shown in Fig. 3, the four edges corresponding to blocks 12, 34, ij and kl form a cycle in G.

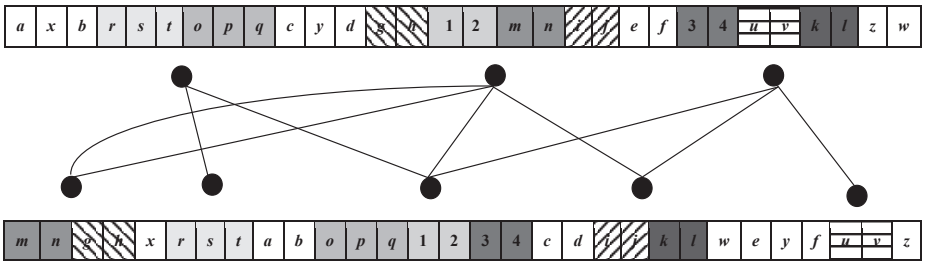

Fig. 3. Multiple edges and cycles in the bipartite graph.

Rule(2.5) Finally, within any super-block, for all blocks between two letters in Σ_1, change each of them into a new letter in Σ_1. For the leftmost (rightmost) super-block in G_1 and G_2, if there is no isolate on its left(right), for all blocks on the left (right) of letters in Σ_1, change each of them into a new letter in Σ_1.

3. Let the resulting sequences be G'_1, G'_2. Return $S \leftarrow \Sigma$ as a parameterized search space.

The correctness of Step 2 is as follows (Lemma 1 covers Rule (2.1)):

Lemma 2. *Rule (2.2) is correct.*

Proof. First, suppose that between P_i, P_j in V_1 there is a P' of length-2 or length-3 which is deleted in some optimal solution. As P' has no isolated neighbor in G_1, deleting it will create a new strip which includes at most two isolated neighbors of it in G_2. Therefore, we can keep P' as a strip and obtain another solution at least as good as the assumed optimal solution (which deletes P').

For P_i and P_j, as they are in V_1 and V_2, each of them has at most 2 isolated neighbors (one each in G_1 and G_2). If some optimal solution deletes one (or both) of them, by the same argument, we can keep one (or both) of them as strips to have a solution at least as good as the assumed optimal solution. □

Note that after Rule (2.2) is run, now a super-block could contain length-2 and length-3 blocks (no two common to another super-block), as well as letters in Σ_1.

Lemma 3. *Rule (2.3) is correct.*

Proof. First, suppose that between P_s, P_t in V_1 (resp. V_2) there is a P'' of length-2 or length-3 which is deleted in some optimal solution. As P'' has no isolated neighbor in G_1 (resp. G_2), deleting it will create a new strip which includes at most two isolated neighbors of it in G_2 (resp. G_1). Therefore, we can keep P'' as a strip and obtain another solution at least as good as the assumed optimal solution (which deletes P'').

For P_s and P_t, each of them has at most 3 isolated neighbors (1 in G_1 and 2 in G_2, or vice versa). If some optimal solution deletes one (or both) of them, by the fact that they are of length-3, we can keep one (or both) of them as strips to have a solution at least as good as the assumed optimal solution. □

After the run of Rule (2.3), a super-block could contain at most one length-3 block, as well as length-2 blocks and, of course, letters in Σ_1.

Lemma 4. *Rule (2.4) is correct.*

Proof. In the simple bipartite block graph G, if there is a cycle, with the involved length-2 or length-3 blocks being $P'_1, P'_2, ..., P'_u$, then $|P'_i| \geq 2$ for $1 \leq i \leq u$. If some optimal solution deletes some of these blocks, say $P'_{i1}, P'_{i2}, ..., P'_{ip}$, then in G we have deleted p edges, each associated with some P'_{ij}. P'_{ij} has at most two isolated neighbors (at most one each in G_1 and G_2). Consequently, we could keep $P'_{i1}, P'_{i2}, ..., P'_{ip}$ as strips to have a solution at least as good as the claimed optimal solution. □

Lemma 5. *Rule (2.5) is correct.*

Proof. In a super-block s_1 in G_1, any block $P' \in s_1$ between two letters in Σ_1 has at most 2 isolated neighbors in G_2. So if some optimal solution deletes P', we can put it back to have a solution at least as good as the assumed optimal solution. □

By now, it is easily seen that any given super-block s, after these run of five rules, has at most two continuous sequences of blocks which are not put in Σ_1. In other words, at this point, each super-block contains at most one letter in Σ_1.

Let Σ_1 be the set of all new letters used in the kernelization process, with $\Sigma_1 \cap \Sigma = \emptyset$. The three lemmas for obtaining the final results are:

Lemma 6. *There is an optimal CMSR solution of size k for G_1 and G_2 if and only if the solution can be obtained by deleting k markers in Σ from G'_1 and G'_2 respectively.*

Notice that after the kernelization step, we have no cycle and no vertex of degree zero in G. So if any connected component in G has q edges, then it must have exactly a set H of $q + 1$ vertices. We have the following lemmas on H.

Lemma 7. *Let G contain m connected components $H_1, H_2, \cdots, H_m$, and let each H_i have q_i edges. Then, in between the vertices in G, there are at least $\sum_{i=1}^{m} q_i + m - 2$ sequences of neighboring isolates in G_1, G_2.*

Proof. The $q_i + 1$ vertices in H_i form a tree. In G_1 and G_2, these vertices correspond to continuous sequences of blocks (each of length at least 2, some of which could have been converted to letters in Σ_1), separated by sequences of neighboring isolates. Let H_i have a_i vertices in G_1 and b_i vertices in G_2. In G_1 the $\sum_{i=1}^{m} a_i$ vertices bound at least $\sum_{i=1}^{m} a_i - 1$ sequences of neighboring isolates. Similarly, in G_2 the $\sum_{i=1}^{m} b_i$ vertices bound at least $\sum_{i=1}^{m} b_i - 1$ sequences of neighboring isolates. In total the vertices in G have bounded at least

$$\left(\sum_{i=1}^{m} a_i\right) - 1 + \left(\sum_{i=1}^{m} b_i\right) - 1 = \sum_{i=1}^{m} q_i + m - 2$$

sequences of neighboring isolates, due to $a_i + b_i = q_i + 1$, for $i = 1..m$. □

Lemma 8. *Given any connected component H in G with q edges, the total length of all the blocks associated with the edges in H is at most $\lceil \frac{5q}{2} \rceil$.*

Proof. It is clear that $3q$ is a trivial upper bound, due to Rules (2.1-2.3). To have this tighter bound, first notice again that the $q + 1$ vertices in H form a tree. Then by the fact that no two incident edges can both correspond to length-3 blocks, we can conclude that the number of length-3 blocks allowed in H is exactly the size of maximum matching of H, which is obviously at most $\lceil q/2 \rceil$ (which occurs when H is in fact a path). Then the total length of all the blocks associated with the edges in H is at most $2q + \lceil q/2 \rceil = \lceil \frac{5q}{2} \rceil$. □

Finally, we have the following theorem.

Theorem 1. *In G'_1 (resp. G'_2), there are at most $18k$ letters (markers) in Σ. In other words, CMSR has a parameterized solution search space of size $18k$.*

Proof. We use an inverse amortized analysis. Assume that we have some optimal MSR solution O^* (i.e., all letters in O^* are in some strips), we try to insert the deleted letters and length-2/3 blocks back into O^* to obtain G_1, G_2. There are four sets of letters/blocks: A — those letters/blocks we insert into G_1, G_2 (of a total length k); B — those isolated letters which were in some strips in O^*, but due to the insertion of type-A letters/blocks, they are broken into isolates; C — those blocks identified by our kernelization algorithm; and D — the remaining length-2/3 blocks associated with the edges in the block graph G. Firstly, we show that

$$|A| + |B| + |D| \leq 18|A| = 18k.$$

Note that although A could contain sequences of blocks, they will be counted into $|A| = k$.

Each inserted type-A letter can break at most two strips in O^*, resulting in at most 4 type-B isolates.

The most general scenario is when we have a graph G each of its vertices corresponds to at most two sequences of type-D blocks, e.g., a vertex in G corresponds to $\mathcal{D} = P_1 P_2 \cdots P_i \cdot \Sigma_1$ letters $\cdot P_{i+1} P_{i+2} \cdots P_l$ (there could be no Σ_1 letters between P_i, P_{i+1}). For this scenario, first recall that now in G we have no

cycle and no vertex of degree zero; moreover, in $\mathcal{D}$ we have at most one length-3 block. So if each connected component $H_i, 1 \leq i \leq m$ in G has q_i edges, then it must have exactly a set of $q_i + 1$ vertices. By Lemma 7, vertices in G bound at least $\sum_{i=1}^{m} q_i + m - 2$ sequences of isolated neighbors.

We now finish the final proof.

First, let us consider the (at least) $\sum_{i=1}^{m} q_i + m - 2$ sequences of isolated neighbors (also called *slots* for convenience) bounded by the vertices of G. These slots are introduced by the insertion of at least $\lceil (\sum_{i=1}^{m} q_i + m - 2)/4 \rceil$ type-A isolates. As what have just been discussed, each of these type-A isolates can introduce at most 4 type-B isolates. By Lemma 8, the total length of all the type-D blocks in G is at most $\sum_{i=1}^{m} \lceil \frac{5q_i}{2} \rceil$. Therefore, each type-A isolate can be charged a total cost of

$$\sum_{i=1}^{m} \lceil \frac{5q_i}{2} \rceil / \lceil \frac{\sum_{i=1}^{m} q_i + m - 2}{4} \rceil + 5,$$

which is at most 18 (when $m = 1$). To see why, let $t = \lceil (\sum_{i=1}^{m} q_i + m - 2)/4 \rceil$. Then

$$\sum_{i=1}^{m} q_i \leq 4t - m + 2.$$

Therefore,

$$\sum_{i=1}^{m} \lceil \frac{5q_i}{2} \rceil \leq \lfloor \frac{5 \sum_{i=1}^{m} q_i}{2} \rfloor + m \leq \lfloor \frac{5(4t - m + 2)}{2} \rfloor + m = 10t + 5 - \lceil \frac{3m}{2} \rceil,$$

which is at most $10t + 3 \leq 13t$, with $m = 1$ and $t \geq 1$. Consequently, this means that our charge is safe.

Second, for each substring of r isolates not bounded (delimited) by vertices of G (we could have at most 4 such substrings of isolates, 2 each at the ends of G_1 and G_2), we first ignore the type-A isolates already contained in some slot and suppose that we have a remaining of r' isolates. These r' isolates can be either of type-A or type-B. It is easy to see that at least $\lceil r'/5 \rceil$ of these remaining r' isolates must be deleted. (The deleted ones are of type-A.) Clearly, $18\lceil r'/5 \rceil > r'$. So again our charge is safe.

As a simple example, assume that $G_1 = \boxed{\text{abc}} w_1 w_2 \boxed{\text{de}} \boxed{\text{fg}} x$ and $G_2 = \boxed{\text{abc}} \boxed{\text{de}} - w_2 x - w_1 \boxed{\text{fg}}$, they form G which contains a single connected component of 4 vertices and 3 edges. We have two slots: $w_1 w_2$ and $-w_2 x - w_1$. As x is charged for a total cost of 18 (including itself), while the length of G_1, G_2 is only 10, so the charge is safe.

Altogether, this gives us an upper bound of $18k$ for $|A| + |B| + |D|$. □

We can show that our kernelization algorithm for constructing the parameterized solution search space S is in fact tight, i.e., the size of S, returned by our algorithm, is at least $18k$ for $k = 1$. It can be done by modifying the $10k$ example at the end of the proof of Theorem 1 as follows.

$G_1 =$ $\boxed{\text{abc}}\,\boxed{\text{de}}\,fxg\,\boxed{\text{hij}}\,\boxed{\text{kl}}\,mn\,\boxed{\text{opq}}$

$G_2 =$ $\boxed{\text{abc}}\,fg\,\boxed{\text{de}}\,\boxed{\text{hij}}\,mxn\,\boxed{\text{kl}}\,\boxed{\text{opq}}$

The corresponding block graph G is a path. The optimal CMSR solution is to delete x (i.e., k=1). So the above parameterized search space bound is in fact tight.

3.2 A Simple Upper Bound for the Kernel of CMSR

We first show an easy upper bound for the kernel of CMSR. To obtain a linear kernel, we need to contract the number of Σ_1 letters while keeping S untouched. Note that, after Rule (2.5), each super-block contains at most one sequence of letters in Σ_1 in both G_1' and G_2'.

Now let us consider the number of super-blocks, each containing at most one sequence of letter in Σ_1. Following the proof of Theorem 1, the number of super-blocks is equal to the number of vertices of G, which is

$$\sum_{i=1}^{m}(q_i+1) = (\sum_{i=1}^{m} q_i) + m \leq (4t - m + 2) + m = 4t + 2,$$

which is bounded by $4k + 2$.

To obtain a kernel, what we can do is simply compressing a maximal sequence of Σ_1 letters (each sequence is called a *long slot* for the ease of presentation) using the following additional rule after Rules (2.1)-(2.5): if a Σ_1 letter which corresponds to an original block $x_1x_2\cdots x_m$ (resp. $-x_m\cdots - x_2 - x_1$) appears leftmost or rightmost in some long slot in either G_1' or G_2', then keep a length-4 block x_1abx_m (resp. $-x_m - b - a - x_1$), where a, b are new letters not in Σ or Σ_1; otherwise, delete it. The correctness of the rule is due to that, while a Σ_1 letter (or, a 4-block) does not have to be deleted some isolate can be connected to the 4-block to form a longer strip in both G_1 and G_2. With our method, on average, we might need to keep four 4-blocks for each long slot in each super-block of $G_i', i = 1, 2$ — the total number of 4-blocks kept is bounded by $(4k + 2) \times 4 = 16k + 8$.

Therefore, we can obtain a kernel for CMSR of size

$$18k \times 2 + (4k + 2) \times 4 \times 4 = 100k + 32 \leq 132k.$$

Suppose that W_j is of length 4 and is disjoint with the letters in W_l, for $j \neq l$, we can modify the $18k$ example to have a tight bound of $132k$ (for $k = 1$) for our method. In the $18k$ example, in the three super-blocks of G_1 we insert three quadruples of blocks $W_1W_2W_3W_4, W_5W_6W_7W_8, W_9W_{10}W_{11}W_{12}$ respectively, in the three super-blocks of G_2 we insert three quadruples of blocks $W_2W_1W_4W_3, W_6W_5W_8W_7, W_{10}W_9W_{12}W_{11}$. After these insertions, the total size of G_1 and G_2 is $18 \times 2 + 6 \times 16 = 132$.

3.3 Computing a Better Linear Kernel for CMSR

The above $132k$ kernel for CMSR can certainly be improved. We need to apply a few different rules.

Rule(2.6): If there exists a 3-block in a super-block which contains some Σ_1 letter, then change this 3-block into a new Σ_1 letter.

Lemma 9. *Rule (2.6) is correct.*

Proof. Since this 3-block Z appears in a super-block (say in G_1) which contains some Σ_1 letter, it has at most one isolated neighbor in G_1 and at most two isolated neighbors in G_2. So if this 3-block was deleted in the optimal solution, there are at most three isolates becoming a part of a strip. (As the super-block in G_1 contains some Σ_1 letter, at most one isolate can contribute to some strip after the deletion of Z.) Therefore, keeping this 3-block Z will result in a solution at least as good as the optimal solution. □

Rule(2.7): If there exists a 3-block in the leftmost (resp. rightmost) super-block in G_1 or G_2, without any isolate on its left (resp. right), then change this 3-block into a new Σ_1 letter.

Lemma 10. *Rule (2.7) is correct.*

Proof. The proof of this lemma is quite the same as that of Lemma 9, hence omitted. □

Finally, we make use of Rule (2.3) to have this rule (2.8) to contract the sequences of blocks corresponding to Σ_1 letters.

Rule(2.8): Suppose that we are given a sequence of Σ_1 letters $\pi_1, \pi_2, \ldots, \pi_j$, where π_i corresponds to some block C_i, $1 \leq i \leq j$. If $j = 1$, which implies that C_1 is of length at least 4 (it can be concluded from Rule (2.2)-(2.5) that a single 2-block or 3-block can not be changed into a Σ_1 letter), then change C_1 into a 4-block by keeping the leftmost and rightmost letters, and keep C_1. If $j > 1$, then change C_1 and C_j into 3-blocks, and keep C_1 and C_j. Finally, delete all the blocks that are not kept in both G_1 and G_2.

Note that, from Lemma 10, we know that while applying Rule (2.8), it is sufficient to keep or construct only one 3-block in the leftmost (resp. rightmost) super-block without any isolate on its left (resp. right) in both G_1 and G_2.

Finally, to obtain the improved kernel, we need to apply the rules in the following order: (2.1),(2.2),(2.3),(2.4),(2.6),(2.7),(2.5),(2.8). On the other hand, we can compute a parameterized search space of size $18k$ by applying rules in the order: (2.1),(2.2),(2.3),(2.4),(2.5).

We thus have the main theorem of this paper.

Theorem 2. *CMSR has a linear kernel of size 78k.*

Proof. Note that the size of a kernel is the total length of both G'_1 and G'_2 after the eight rules are applied. As what has been discussed in Theorem 1, the total number of letter in type-A and type-B is at most $5k$. From Theorem 1, the number of super-blocks is at most $4k+2$, hence the number of type-D blocks is at most $4k+1$. From Lemma 10 and Rule (2.8), there are at most $((4k-2)\times 2+4)\times 2$ 3-blocks that were kept or constructed after applying Rule (2.6) and (2.8). From Lemma 9, the total number of letters in the type-D blocks is at most $(4k+1)\times 2$. So the size of kernel is bounded by

$$5k \times 2 + ((4k-2) \times 2 + 4) \times 2 \times 3 + (4k+1) \times 2 \times 2 = 74k + 4 \leq 78k. \quad (1)$$

□

Corollary 1. *Combined with the bounded search tree method, CMSR can be solved in $O(2.36^k k^2 + n^2)$ time.*

Proof. Without the linear kernel bound, using the bounded search tree method, there is an FPT algorithm which runs in $O(2.36^k n^2)$ time [3]. With the $78k$ linear kernel, the running time of the corresponding algorithm can be improved to $O(2.36^k k^2 + n^2)$ time. This is a standard procedure: just run the algorithm on the linear kernel. □

We comment that, by modifying the example at the end of Section 3.1, we can obtain a linear kernel of size $78k$ (for $k = 1$). This shows that the $78k$ kernel bound for CMSR is tight for the method (at least for $k = 1$).

G_1= [123] [456] [ab] [de] fzg [hi] [$y_1y_2y_3$] [$x_4x_5x_6$] [$x_7x_8x_9$] [$y_4y_5y_6$] [kl] mn [op] [789] [$x_1x_2x_3$]

G_2= [456] [123] [ab] fg [de] [$x_4x_5x_6$] [$y_1y_2y_3$] [$y_4y_5y_6$] [$x_7x_8x_9$] [hi] mzn [kl] [op] [$x_1x_2x_3$] [789]

4 Concluding Remarks

We show a non-trivial $78k$ linear kernel for the Complementary Maximal Strip Recovery problem. Combined with a known bounded search tree algorithm, this results in the best known FPT algorithm for CMSR — in $O(2.36^k k^2 + n^2)$ time. An interesting question is whether these bounds can be further improved.

Using the recent concept of weak kernels [18], Theorem 2 in fact implies that CMSR has a (direct) weak kernel of size $18k$. However, as direct weak kernels can all be transformed into the traditional kernels (from the experience as in this paper), we think it is better to use weak kernels solely for the indirect ones. For problems admitting linear indirect weak kernels (e.g., Sorting by Reversals [18] and Sorting by Unsigned DCJ Operations [19]), no linear/polynomial kernels are known and no known bounded search tree algorithm can match up with the solutions provided by weak kernels.

Acknowledgments

This research is partially supported by NSF grant DMS-0918034, by NSF of China under grant 60928006 and 61202014, by the Open Fund of Top Key Discipline of Computer Software and Theory in Zhejiang Provincial Colleges at Zhejiang Normal University, by China Postdoctoral Science Foundation funded project under grant 2011M501133 and by the Shanghai Thousand Talents Program. We also thank the anonymous reviewers for several useful comments.

References

1. H. Bodlaender, R. Downey, M. Fellows and D. Hermelin. On problems without polynomial kernels. In *Proc. 35th Intl. Colloquium on Automata, Languages and Programming (ICALP'08), pages 563-574, 2008.*
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. L. Bulteau, G. Fertin, M. Jiang and I. Rusu. Tractability and approximability of maximal strip recovery. *Theoretical Computer Science*, 440-441:14-28, 2012.
4. Z. Chen, B. Fu, M. Jiang, and B. Zhu. On recovering syntenic blocks from comparative maps. *Journal of Combinatorial Optimization*, 18(3):307–318, 2009.
5. V. Choi, C. Zheng, Q. Zhu, and D. Sankoff. Algorithms for the extraction of synteny blocks from comparative maps. In *Proceedings of the 7th International Workshop on Algorithms in Bioinformatics (WABI'07)*, pages 277-288, 2007.
6. S. Cook. The complexity of theorem-proving procedures. In *Proceedings of the 3rd ACM Symp. on Theory of Computing (STOC'71)*, pages 151-158, 1971.
7. R. Downey and M. Fellows. *Parameterized Complexity*, Springer-Verlag, 1999.
8. H. Dell and D. van Melkebeek. Satisfiability allows no nontrivial sparsification unless the polynomial-time hierarchy collapses. In *Proc. 42nd ACM Symp. Theory of Computation (STOC'10), pages 251-260, Cambridge, MA, USA, 2010.*
9. M. Fellows. The lost continent of polynomial time: preprocessing and kernelization. In *Proc. 2nd Intl. Workshop on Parameterized and Exact Computation (IWPEC'06)*, LNCS 4169, pages 276-277, 2006.
10. H. Fernau, F. Fomin, D. Lokshtanov, D. Raible, S. Saurabh and Y. Villanger. Kernel(s) for problems with no kernel: on out-trees with many leaves. In *Proc. 26th Intl. Symp. on Theoretical Aspects of Computer Science (STACS'09)*, pages 421-432, 2009.
11. J. Flum and M. Grohe. *Parameterized Complexity Theory*, Springer-Verlag. 2006.
12. L. Fortnow and R. Santhanam. Infeasibility of instance compression and succinct PCPs for NP. In *Proc. 40th ACM Symp. Theory of Computation (STOC'08), pages 133-142, Victoria, Canada, 2008.*
13. M. R. Garey and D. S. Johnson. *Computers and Intractability: A Guide to the Theory of NP-Completeness.* W. H. Freeman, 1979.
14. J. Guo and R. Niedermeier. Invitation to data reduction and problem kernelization. *SIGACT News*, 38:31-45. 2007.
15. 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.

16. 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.
17. H. Jiang, Z. Li, G. Lin, L. Wang and B. Zhu. Exact and approximation algorithms for the complementary maximal strip recovery problem. *J. of Combinatorial Optimization*, 23(4):493-506, May, 2012.
18. H. Jiang, C. Zhang and B. Zhu. *Weak Kernels*. ECCC Report, TR10-005, Oct, 2010.
19. H. Jiang, B. Zhu and D. Zhu. Algorithms for sorting unsigned linear genomes by the DCJ operations. *Bioinformatics*, 27:311-316, Feb, 2011.
20. Z. Li, R. Goebel, L. Wang and G. Lin. An improved approximation algorithm for the complementary maximal strip recovery problem. *Proceedings of the 2001 Joint FAW-AAIM Conf (FAW-AAIM'01)*, LNCS 6681, pages 46-57, 2011.
21. R. Niedermeier. *Invitation to Fixed-Parameter Algorithms*, Oxford Univ. Press. 2006.
22. R. Karp. Reducibility among combinatorial problems. In R. Miller and J. Thatcher (eds.), *Complexity of Computer Computations*, Plenum Press, NY, pages 85–103, 1972.
23. L. Wang and B. Zhu. On the tractability of maximal strip recovery. *J. of Computational Biology*, 17(7):907-914, 2010. (Correction: 18(1):129, Jan, 2011.)
24. 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.