Electronic Colloquium on Computational Complexity, Report No. 95 (2009)

Strip Exchanging is Hard

Swapnoneel Roy*

State University of New York at Buffalo,
Buffalo, New York 14220, United States
http://www.cse.buffalo.edu/

Abstract. The sorting by strip exchanges (aka strip exchanging) problem is motivated by applications in optical character recognition and genome rearrangement analysis. While a couple of approximation algorithms have been designed for the problem, nothing has been known about its computational hardness till date. In this work, we show that the decision version of strip exchanging is NP-Complete. We present a few new lower bounds for the problem along with the stated result. The NP-Completeness proof is based on reducing sorting by strip moves, an already known NP-Complete problem, to strip exchanging. The former is a widely studied problem also motivated by optical character recognition and genome rearrangement analysis.

Introduction

0.1 Motivation

The *sorting by strip exchanges* problem is minimizing the number of strip exchanging moves to sort a permutation. A strip exchanging move constitutes interchanging the positions of two *strips* in the permutation. A *strip* is a maximal sorted sub-string of the initial permutation. As an example, in the permutation 8 2 5 6 3 9 1 4 7 on nine elements, there are eight strips, and {5 6} is the only strip containing more than a single element. Strips once formed are not broken in subsequent steps, but are joined to form larger strips. The final permutation is the sorted or identity permutation, which contains only one strip. As an example, the permutation 1 4 3 2 5 could be sorted to 1 2 3 4 5 by using a single strip exchanging move: exchange 4 and 2. Without loss of generality, the strips of any permutation could be replaced by just a single element.
The sorting by strip exchanges problem is motivated by its application in the area of optical character recognition. In optical character recognition, certain text regions are marked and referred to as *zones*. Its essential to have all the zones in the correct order in the final output. This could be achieved by a strip exchanging algorithm. In this aspect, its quite similar to the problem of *sorting*

* Department of Computer Science and Engineering, State University of New York at Buffalo, Buffalo, NY.

ISSN 1433-8092

by strip moves [1] [1], [2], [3], [4], [5]. The sorting by strip moves problem is minimizing the number of strip moves to sort a permutation. A strip move is just moving a strip to a different position in the permutation. As an example, the same permutation 1 4 3 2 5 could be sorted to 1 2 3 4 5 by using 2 strip moves: move 4 and move 2. Bein *et al* [3] showed that the problem of getting the zones in order is essentially equivalent to sorting by strip moves. From now on, we would refer sorting by strip moves simply by strip moves, and sorting by strip exchanges simply by strip exchanging in this paper. Both the problems strip moves and strip exchanges are combinatorial optimization problems which optimize the number of steps to sort a given permutation π into the identity permutation *id*. This sorting essentially involves reducing the number of strips s in π to 1 in *id*. So algorithms designed for both the problems should target to reduce the number of strips by the maximum possible at each step.

Its easy to note that a single strip move can reduce the number of strips by at most 3 at each step. In contrast, a single strip exchanging move can reduce the number of strips at most by 4. Since the maximum number of strips that can reduced by a strip exchanging move is *more* than that of a strip move, algorithms for the strip exchanging problem are expected to perform *better* than those for strip moves for the optical character recognition problem. A little insight into the problem of strip moves will reveal the fact that a strip move in fact also *interchanges* the positions of *two* sub-strings in π. One is the strip being moved, and the other is its adjacent sub-string, which might or might not be a strip. Hence the cost of a strip move operation is essentially *equal* to the cost of a strip exchanging operation, which also involves interchanging the positions of two sub-strings in π. Hence its really interesting to know about the computational hardness of the strip exchanging problem and have efficient algorithms designed for it. The strip move problem has been shown to be NP-Complete and the best known algorithm for it is a 2-approximation. Anything about the computational hardness of strip exchanging has not been known till date, and the best known algorithm for it is a 2-approximation.

The strip exchanging problem falls under the category of a class of problems called the genome rearrangement problems. Over the last decade, sorting under different such primitives like reversals [8], [9], [10], [11], transpositions [12], [13], [14], strip moves [1], [2], [3], [4], [5], and block interchanges [6], [7] have been extensively studied. While sorting under reversals [8] has been proved to be computationally hard, a big open problem for over a decade is the computational complexity of sorting by transpositions.

One way to define transposition is, it involves interchanging the positions of any two *adjacent sub-strings* of *any length* in the given permutation π. A block

[1] The sorting by strip moves problem [1], [2] is also known as block sorting [3], [4], [5] in the literature. In order to avoid confusion with another problem, sorting by block interchanges, we use the term strip moves for block sorting in this paper.

interchange is a generalization of a transposition. In a block interchange the positions of *any two non-intersecting sub-strings* of any length are interchanged in π. Sorting by block interchanges has been shown to be polynomially solvable [6]. Another way to define transposition is, it involves moving a sub-string of any length to a different position in π. According to this definition, a strip move is a restricted case of transposition. A strip move a transposition in which the sub-string moved is a strip. Sorting by strip moves has been proved to be NP-Complete [3]. Our primitive at hand, the strip exchange, is a restricted case of block interchanges. A strip exchange is essentially a block interchange where the sub-strings interchanged are strips. The study of the computational complexity of strip exchanging may give further insight into the computational complexity of transpositions, which is a very important open question in genome rearrangement analysis. A general case of transpositions, the block interchanges is polynomial, while a restricted case of it, the strip moves is NP-Complete. Strip exchanging, like transpositions, lies somewhere in between block interchanges and strip moves. Hence knowing about its complexity is definitely going one step ahead to knowing about the computational complexity of transpositions. Moreover, as with the other primitives, it is interesting to study strip exchanging, and develop new algorithms for it to apply in the genome rearrangement analysis.

0.2 Overview of the result and techniques

The sorting by block interchanges problem has been shown to be polynomially solvable by an algorithm called the minimal block interchanges algorithm [6]. Given an unsorted permutation π, we note that there must be at least two elements x and y in π which are in *wrong order*. That is $x < y$ but $\pi = [1 \dots y \dots x \dots]$. The algorithm chooses x and y to be the minimum and maximum such values respectively. Thus, by this choice, we have the element $x-1$ to the left of y and the element $y+1$ to the right of x in π. Hence π is like $\pi = [1 \dots x-1 \dots y \dots x \dots y+1 \dots]$. Now interchanging the sub-strings starting from the element exactly to the right of $x-1$ till y and starting from x till the element exactly to the left of $y+1$ by a block interchanging move gives us the permutation $\pi' = [1 \dots x-1 x \dots y y+1 \dots]$. So by this move, we have at least brought a pair of elements in order. If two elements are adjacent in π and are not adjacent in the sorted or identity permutation id, those two elements form a *break-point* in π. By performing the move mentioned here, we have essentially removed at least 2 breakpoints from π. This has been showed to be optimal in [6]. We note that id does not contain any break-point.

Note that the removal of 1 break-point corresponds to decreasing the number of strips in π by 1. We remark that the algorithm minimal block interchanges might not work for strip exchanges. Notice that the two sub-strings which the algorithm interchanges might always not be strips. If they are not strips, the move would not be permitted by the primitive strip exchanges.

A *reversal* in π is a pair of consecutive elements ab such that $a > b$. While reducing strip moves to strip exchanging, we first construct a red-blue graph $G(\pi)$ for π, as described in [3]. Given π, we draw the *blue edges* between each pair of reversals. The *red edges* are drawn between consecutive elements forming a *maximal increasing sub-sequence* in π. If there are more than one such sub-sequence, we choose any one of them. As an example. for $\pi = 4\ 3\ 1\ 9\ 5\ 2\ 11\ 8\ 6\ 10\ 7\ 12$, the reversals are $(4,3)$, $(3,1)$, $(9,5)$, $(5,2)$, $(11,8)$, $(8,6)$ and $(10,7)$. Hence we have 7 blue edges in $G(\pi)$. There are several maximal increasing sub-sequences in π. Some of them are: $4-5-8-10-12$, $3-5-8-10-12$, $1-5-8-10-12$, $1-2-8-10-12$ and, $1-2-6-7-12$. If we choose any one of them, we would have 4 red edges in $G(\pi)$. In [3], the intuition behind drawing the red edges has been shown. We would illustrate it briefly in section 1. Further it was shown in [3] that $G(\pi)$ cannot have cycles (over both kinds of edges). In [3] a lower bound for strip moves was identified as the number of blue edges in $G(\pi)$. This lower bound was shown to be tight for π *iff* $G(\pi)$ is a tree. To be more precise, iff $G(\pi)$ is connected. We identify a new lower bound for strip exchange here as *half the number of blue edges* in $G(\pi)$. Next we note that $G(\pi)$ being a tree is a *necessary but not a sufficient* condition for a strip exchanging schedule to exist on π, which matches this lower bound.

In the reduction, an arbitrary instance of strip moves is reduced to a specific instance of strip exchanging. The reduction procedure goes on like this: Given any arbitrary permutation π, we try to construct a longer permutation π' by inserting at most $b(\pi)$ elements in π, where $b(\pi)$ is the number of blue edges in $G(\pi)$. Our insertion procedure meets the following conditions:

- If $G(\pi)$ is connected, then we have an optimal strip moves schedule for π. In this case, $G(\pi')$ remains connected after the procedure and we have an optimal strip exchanging schedule on π'.
- If $G(\pi)$ is not connected, then we do not have an optimal strip moves schedule for π. In this case, $G(\pi')$ remains disconnected after the procedure is performed. Hence we do not have an optimal strip exchanging schedule on π'.

The idea behind the reduction procedure is to have *two* reversals in π' (blue edges e_1, e_2 in $G(\pi')$), for each reversal in π (blue edge e in $G(\pi)$) such that we have a strip moves move to eliminate e from $G(\pi)$ *iff* we have a strip exchanging move to eliminate e_1 and e_2 from $G(\pi)$. The steps following by the procedure are as follows:

- Input: An arbitrary permutation π.
- Output: A specific permutation π'.
- Construct red-blue graph $G(\pi)$ from π.
- For each blue edge e.
  - Insert an element i in π such that it creates another blue edge e' in π' and we have a strip exchanging move to eliminate both e and e'.

– Graph $G(\pi')$ for the reduced permutation π' is obtained at the end of the reduction.

There are some cases in the above procedure where a slight variation of the above steps are followed. But nevertheless, we note that the maximum number of elements inserted in π would by the above reduction procedure would be $b(\pi)$ (one for each blue edge in $G(\pi)$). Also each new element inserted would add exactly *one* new blue edge in $G(\pi)$. Now let $b(\pi)$ and $b(\pi')$ are the number of blue edges in $G(\pi)$ and $G(\pi')$ respectively. Then the above reduction procedure guarantees that there exists a strip moves schedule for π of $b(\pi)$ steps *iff* there exists a strip exchanging schedule for π' of $b(\pi')/2$ steps. We note that $b(\pi)$ and $b(\pi')/2$ are lower bounds for strip moves and strip exchanging respectively. Hence the reduction procedure guarantees that there exists an optimal strip moves schedule for π *iff* there exists an optimal strip exchanging schedule for π').

The reduction outlined above proves that the problem of strip exchanging is NP-Complete. In Section 1, we present various lower bounds for the problem. In Section 2, we show the decision version of the problem to be NP-Complete by reducing a known NP-Complete problem, strip moves (aka block sorting) to it.

1 Lower Bounds for Strip Exchanging

A strip exchanging schedule $se(\pi)$ for π is a shortest sequence of strip exchanging moves to sort π. The decision version of the strip exchanging problem would be, given a permutation π and an integer $k > 0$, is $se(\pi) = k$? In other words, can we find a strip exchanging schedule of length k? It is trivial to observe that strip exchanging belongs to NP. But the open question was whether it was NP-Complete or polynomially solvable.
In section 0.2, we observed that a lower bound for this problem is $se(\pi) \geq \lceil (s-1)/4 \rceil$, where s is the number of strips in π. We illustrated the concept of a *red-blue* graph $G(\pi)$ for a permutation π in section 0.2. Here we use this concept to get a new lower bound for strip exchanging. Later on, the concept of this graph has been used in the hardness reduction. We now formally define and illustrate a few terms from [3] which have already been introduced in section 0.2.

Definition 1 (Reversals). *In π, a reversal is a pair of consecutive elements ab such that $a > b$.*

Definition 2 (Maximal increasing sub-sequence). *In π, a longest increasing sub-sequence is a maximal sequence of elements which are monotonically increasing.*

We now state a prove a new lower bound for strip exchanging.

Lemma 1. *Let $rev(\pi)$ be the number of reversals in π. Then $se(\pi) \geq \lceil rev(\pi)/2 \rceil$.*

Proof. It is easy to see that one strip exchanging move can reduce $rev(\pi)$ at most by 2. Since id does not contain any reversals, strip exchanging is simply reducing $rev(\pi)$ to 0. Hence it would require at least $\lceil rev(\pi)/2 \rceil$ strip exchanging moves to sort π. Hence lemma 1 holds. □

We denote the position of element a in π as $\pi(a)$. Note that all the strips in π could be replaced by a single element without loss of generality. This is because we do not *break* any strip in a strip move or a strip exchanging schedule. When all the strips in π is replaced by a single element, it is termed as *reduced permutation* [3] or a *kernel permutation* [5]. Now onwards we assume π to be reduced or kernel w.l.o.g. We observe in a short while that this also does not change the number of blue or red edges in the red-blue graph $G(\pi)$ constructed from π. Given π, we construct the red-blue graph $G(\pi)$ in the following way:

- A **blue edge** is constructed between the participating elements a and b of each reversal (a, b).
- A **red edge** is constructed between two elements a and b if a and b are *consecutive elements* of a maximal increasing sub-sequence of π.

The intuition behind the red edges is to treat the elements (strips) participating as already in their correct positions [3]. We need to move only the other elements in the strip move or strip exchanging schedule. Thus we effectively *save* a few moves here. Now we present a few properties about the red edges proved in [3]. We would state them here as lemmas but would not prove them.

Lemma 2. *[3] A red edge is constructed between elements a and b in π if:*

- *$a < b$ and $\pi(a) < \pi(b)$ and*
- *a and b are joined to form a strip before either is moved and*
- *If $\pi(a) < \pi(c) < \pi(b)$, then c is moved before a and b are joined.*

Lemma 3. *[3] Any node x can have a red degree of at most 2. One from y to x where $y < x$ another from x to z where $x < z$.*

Lemma 4. *[3] For any π, $G(\pi)$ is acyclic over both red and blue edges.*

A **perfect strip moves** schedule [3] on π is a strip moves schedule which sorts π in $rev(\pi)$ moves. Note that $rev(\pi)$ is equal to the number of blue edges in graph $G(\pi)$.

Lemma 5. *[3] There exists a perfect strip moves schedule on π iff $G(\pi)$ is a tree.*

An **exact strip exchanging** schedule on π is a strip exchanging schedule which sorts π in $\lceil rev(\pi)/2 \rceil$ moves. Both perfect strip moves and exact strip exchanging are optimal.

Lemma 6. *The red-blue graph $G(\pi)$ should be a tree in order for an exact strip exchanging schedule on permutation π to exist.*

Proof. By Lemma 4, $G(\pi)$ cannot have any cycles implies, it could only be a tree, or a forest. Now when $G(\pi)$ is a forest, let it have two components C_1 and C_2. Since $G(\pi)$ is not connected, at a stage of any strip exchanging schedule, we would have a vertex which is isolated. This happens after all the elements in the connected components have been joined to form strips. We note that we cannot find any strip exchanging move to move the strip in an isolated vertex which decreases the number of blue edges by 2. This is because, the strip (element) of the isolated vertex does not participate in a reversal, and hence is not connected to any vertices with a blue edge. We could reduce the number of blue edges at most by 1 by performing a strip exchanging move involving that element. So we clearly need more moves than $\lceil rev(\pi)/2 \rceil$ to sort π by strip exchanging, which is not exact. □

Its important to note that Lemma 5 is a necessary and sufficient condition for having a perfect strip moves schedule on π [3]. In case of a exact strip exchanging schedule, its a necessary, but by no means a sufficient condition. This is because, the lower bound for strip exchanging with respect to the number of reversals (blue edges in $G(\pi)$) is approximately just half of the lower bound for strip moves.

2 The Strip Exchanging Problem is NP-Complete

In this work, we reduce strip moves to strip exchanging in this way: Given any arbitrary instance of strip moves, a permutation π, we construct a specific instance of strip exchanging, another permutation π' by adding a few elements to π. We then show that a perfect strip moves schedule would exist for π iff there exists an exact strip exchanging schedule for π'.

2.1 In NP

First, its easy to see that strip exchanging belongs to NP. A polynomial-time non-deterministic Turing machine can always be designed that simply guesses a schedule of k exchanges and then deterministically checks whether the schedule when applied on the given permutation π, results in id.

2.2 Construction of π' from π

The idea is for each blue edge b in π, we try to insert an element e in π such that it creates a new blue edge b' in π'. And we have a strip exchanging move to simultaneously remove b and b' from π'.We introduce a few terms used in the reduction.
A *blue (red) chain* in the red-blue graph $G(\pi)$ is a maximal sequence of adjacent elements connected by blue (red) edges. As an example the sequence 7 5 4 2 in the permutation 1 7 5 4 2 6 3 forms a blue chain. For any element i in π, $prev(i) = i - 1$ and $next(i) = i + 1$.For the smallest element i in id (and π), we define $prev(i) = 0$ and for $i = n$, we define $next(i) = \infty$, where n is the number of elements in π (and id).

Definition 3 (Move element). *For each blue edge b in $G(\pi)$, a move element i is an element in π which:*

- *is one of the end vertices of b and*
- *has no red degree and*
- *is not a move element for any other blue edge b'.*

If both the vertices x and y to which b is adjacent, have no red degree, and is not a move element for any other blue edge b', then the move element for b is the smaller element among x and y. A variable $move(i)$ associated with every move element i in π. Initially $move(i) = 0$ for all the move elements. It gets updated for two elements in the resulting permutation, after each step of execution of the reduction procedure. A free *element during any step in the reduction is a move element for which $move(i) = 0$. Note that both x and y cannot have a red degree.*

We would have the following variables and subroutines in the reduction procedure:

- We consider blue edge $b = (x, y)$ (clearly $x > y$) with the *least free element either x or y* at each step of execution of the reduction procedure.
- Clearly, the least free element at this step (either x or y) is the move element of blue edge (x, y).
- Three *pivotal elements* $p1$, $p2$, and $p3$. They, along with a few other elements compute the value of the new element e and its position to be inserted at each step.
- Subroutines findPivotalLess(x, y) and findPivotalMore(x, y) called when y is least free element and x is least free element respectively, determine the pivotal elements $p1$, $p2$, and $p3$ at each step.
- Subroutines placeInFront(x, y, $p1$, $p2$, $p3$) and placeAtBack(x, y, $p1$, $p2$, $p3$) called when y is least free element and x is least free element respectively, determine the value and position of the new element e to be inserted.

We describe the main reduction procedure in Algorithm 1. Two subroutines findPivotalLess and placeInFront used in the main reduction procedure, are described in Algorithms 2 and 3 respectively.

Input: Any arbitrary permutation π from any given instance of strip moves
Output: Permutation π' for a specific instance of strip exchanging
Construct red-blue graph $G(\pi)$ and mark the move elements for each blue edge in $G(\pi)$;
foreach *Move element* i *in* π **do**
  Set $move(i) = 0$;
end
while *There are unmarked blue edges in* $G(\pi)$ **do**
  Let $b = (x, y)$ be the blue edge under consideration;
  /* b has least free element in π which might be x or y */
  /* x is the larger element in b that is, $x > y$ */
  if y *is least free element* **then**
    findPivotalLess(x, y);
    Let $p1$, $p2$, and $p3$ be the three pivotal elements;
    placeInFront(x, y, $p1$, $p2$, $p3$);
  end
  else
    /* x is least free element */
    findPivotalMore(x, y);
    Let $p1$, $p2$, and $p3$ be the three pivotal elements;
    placeAtBack(x, y, $p1$, $p2$, $p3$);
  end
end
Permutation π' is obtained from constructed red-blue graph $G(\pi')$;

Algorithm 1: The reduction procedure

The other two subroutines findPivotalMore(x, y) and placeAtBack(x, y, $p1$, $p2$, $p3$) are very similar and omitted here due to space constraints. We include it in the appendix section however. This finishes the construction of permutation π', given π. Please see Figure 1 in the appendix for an example. Here the reduction procedure function f is defined for all arguments π by $f(\pi) = \pi'$. We prove the correctness and complexity of this reduction procedure in Section 2.3 and Section 2.4 respectively.

Input: The elements x and y
Output: The pivotal elements $p1$, $p2$, and $p3$
`/* This subroutine is called when` y `is least free element */`
`/* Decide` $p1$ `*/`
if *move*(x) *does not exist, or move*(x) *is equal to* 0 **then**
  $p1 = x$;
end
else
  `/* This case arises if` x `has been inserted at a previous step of the reduction. */`
  Start from x and traverse the chain containing x and y towards left;
  Stop at element w, where w is either not a move element, or not a free element;
  $p1 = w$;
end
`/* Decide` $p2$ `*/`
if y *is the rightmost element of the chain containing* y **then**
  $p2 = next(y)$
end
else
  Let r be the rightmost element of the chain containing y;
  if *move*(r) *is equal to* $next(y)$ **then**
    $p2 = next(next(y))$;
  end
  else
    $p2 = next(y)$;
  end
end
`/* Decide` $p3$ `*/`
if z *is just before* $p2$ *and there exists a blue edge between* z *and* $p2$ **then**
  $p3 = z$;
end
else
  $p3 = 0$;
end

Algorithm 2: The findPivotalLess(x, y) subroutine

2.3 Complexity

It is not hard to see that f is computable in polynomial time. Constructing $G(\pi)$ takes $O(n^2)$ time, setting $move(i)$ for all the move elements takes $O(n)$. Each

subroutine in the while loop takes $O(n)$ time; so the loop takes $O(n^2)$ time to execute. This makes the total run-time of f equal to $O(n^2)$.

Input: The elements x and y and the pivotal elements $p1$, $p2$, and $p3$
Output: A new element e to be inserted and its position to be inserted based on some cases

if $p3$ *is equal to* 0 **then**
  e is an element with a value between $p1$ and $next(p1)$ i.e $p1 < e < next(p1)$;
  Insert e just in front of $p2$ (adjacent to $p2$);
  Set $move(y)$ to e and $move(e)$ to y ;
  Mark blue edges (x, y) and $(e, p2)$;
end
else
  `/* p3 is not equal to 0 */`
  if $p3$ *is free and is equal to* $next(p1)$ **then**
    `/* This is the only case in which we do not need to insert any new element */`
    Set $move(y)$ to $p3$ and $move(p3)$ to y;
    Mark blue edges (x, y) and $(p3, p2)$;
  end
  if *(*$p3$ *is greater than* $next(p1)$*) or (*$p3$ *is not free and is equal to* $next(p1)$*)* **then**
    e is an element with a value between $p1$ and $next(p1)$ i.e $p1 < e < next(p1)$;
    Insert e between $p3$ and $p2$;
    Set $move(y)$ to e and $move(e)$ to y;
    Mark blue edges (x, y) and $(e, p2)$;
  end
  if $p3$ *is less than* $p1$ **then**
    e is an element with a value between $prev(p3)$ and $p3$ i.e $prev(p3) < e < p3$;
    Insert e right after p1 i.e at position $\pi(p1) + 1$;
    Set $move(y)$ to $p3$ and $move(p3)$ to y;
    if $p1$ *is not equal to* x **then**
      Mark blue edges (x, y) and $(p3, p2)$;
    end
    else
      Mark blue edges (e, y) and $(p3, p2)$;
    end
  end
end

Algorithm 3: The placeInFront(x, y, $p1$, $p2$, $p3$) subroutine

2.4 Correctness

We need to show here that a perfect strip moves schedule exists for π *iff* an exact strip exchanging schedule exists for π'. For that we prove the following lemmas.

Lemma 7. *A new element e, whenever added to π at any step in the reduction procedure f, creates and adds exactly 1 blue edge in $G(\pi)$.*

Proof. We consider each of the 3 cases of element insertion of the subroutine placeInFront (Algorithm 3) and show that in each of these cases exactly one blue edge is created and added to $G(\pi)$. The arguments for the subroutine placeAtBack is analogous and is omitted here. For a recap, we call placeInFront when we consider a blue edge (x, y), with $x > y$, and y is the least free element.

- **Case 1 ($p3 = 0$):** We have $p1$ equal to x or w here. Clearly, $w > x$ since it lies to the left of x in a chain. Again $p2$ equals $next(y)$ or $next(next(y))$. We take $p1 = x$, and $p2 = next(y)$ for the sake of simplicity. We however note the claim holds for the other values of $p1$ and $p2$ as well.
  Since $x > y$, and $next(y) > y$, $p1 \geq p2$. Now we have $p1 < e < next(p1)$. Hence $e > p2$ and therefore $(e, p2)$ is a new blue edge in $G(\pi)$. Moreover, if there was an element z right in front of $p2$, and $p2$ and z did not form a blue edge, we have $z < p2$. Hence $z < e$, and z and e do not form a blue edge in $G(\pi)$. Therefore only one new blue edge $(e, p2)$ is added to $G(\pi)$. If $p1 = w > x$, and $p2 = next(next(y))$, we clearly have $p1 > p2$, since $p1 \neq p2$. Hence the lemma holds.
- **Case 2 ($p3 \neq 0$):** In this case $(p3, p2)$ is a blue edge in $G(\pi)$. Clearly $p3 < p2$. There are two sub-cases in which a new element e gets added. We omit the sub-case in which no element gets added, because there is nothing to prove.
  - $p3 > next(p1)$**:** Just like above, lets take $p1 = x$, and $p2 = next(y)$. Here we insert element e where $p1 < e < next(p1)$ between $p3$ and $p2$. Since $p3 > next(p1)$, we have $p3 > e$, and thus $(p3, e)$ forms a blue edge in $G(\pi)$. Again $e > p1$, and $p1 \geq p2$. Hence $e > p2$. Hence $(e, p2)$ forms a blue edge. Here due to this insertion, blue edge $(p3, p2)$ is eliminated, and two new blue edges $(p3, e)$, and $(e, p2)$ are added to $G(\pi)$. Hence effectively a single new blue edge gets added. The arguments for the other case where $p1 = w > x$ and $p2 = next(next(y))$ are very similar.
  - $p3 < p1$**:** In this case we insert $prev(p3) < e < p3$ right after $p1$. Since $e < p3 < p1$, $(p1, e)$ forms a new blue edge. Now if we take $p1 = x$ and $p2 = next(y)$, then since $p3 > p2$, we have $e > p3 > y$. Hence (e, y) forms another new blue edge. Therefore in this case, blue edge (x, y) gets eliminated by the insertion, and two new blue edges (x, e) and (e, y) get added to $G(\pi)$. Hence a single blue edge is inserted in effect. The case where $p1 = w > x$ would follow by a very similar argument. The case where w does not form a blue edge with the element u to its right (if u exists) is trivial. In the other case, (w, u) is a blue edge with $w > x$ and $u > y$, since it lies to the left of blue edge (x, y) in the same chain. The argument in this case is very similar to the above.

□

Observation 1 *A new element e, whenever added to π at any step in the reduction procedure f, does not change the number of red edges in $G(\pi)$.*

Corollaries 1 and 2 immediately follow from Lemma 7 and Observation 1.

Corollary 1. *The number of new blue edges inserted to $G(\pi)$ during the reduction, is equal to the number of new elements inserted to π.*

Corollary 2. *If $G(\pi)$ is a tree, $G(\pi')$ is also a tree.*

Lemma 8. *If $G(\pi')$ is a tree, $G(\pi)$ is also a tree.*

Proof. In case $G(\pi)$ was not a tree, it was disconnected. This implies there is at least two components C_1 and C_2 in $G(\pi)$. Now suppose x new elements get inserted to π to form π' during the reduction. By corollary 1 the number of blue edges which gets added to $G(\pi)$ to form $G(\pi')$ is x. But at least $x + 1$ edges are required to connect C_1 and C_2 along with the x new elements. Hence $G(\pi')$ cannot be connected too. Hence the lemma. □

Lemma 7 implies, in case we have a perfect strip moves schedule on π, we could have an exact strip exchanging schedule in π'. Lemma 7 was important to prove because had $G(\pi')$ got disconnected due to the procedure, then by lemma 6, we could not obtain an exact strip exchanging schedule for π'.

Lemma 9. *If there exists a perfect strip moves schedule for π, then there exists an exact strip exchanging schedule for π'.*

Proof. We note that the following algorithm would give us an exact strip exchanging schedule for π':

1. Pick the least unsorted element i in π'
2. Exchange the strip s_1 containing element i with the strip s_2 containing $move(i)$
3. Return to first step while there are blue edges left in $G(\pi')$

Note that the above algorithm eliminates 2 blue edges from $G(\pi')$ at every step. Eliminating the 2 blue edges do not affect other edges or add any new edge since both the elements interchanged join with some other strip in π'. The number of strips reduces by at least 2 at each step. Hence it gives us an exact strip exchanging schedule. □

Lemma 10. *If there exist an exact strip exchanging schedule for π', then there exist a perfect strip moves schedule for π.*

Proof. If there exists an an exact strip exchanging schedule for π', then $G(\pi')$ must be a tree. Then by Lemma 8, $G(\pi)$ is also a tree. Hence by Lemma 5, there is a perfect strip moves schedule for π. □

Lemma 9 and Lemma 10 implies that a perfect strip moves schedule exists for π *iff* an exact strip exchanging schedule exists for π'. This leads to the main result.

Theorem 1. *Strip exchanging is* NP*-Complete.*

3 Conclusion

We have shown the strip exchanging problem to be NP-Complete here. This was interesting because sorting under exchanging any two sub-strings and not just strips has been shown to be polynomial. This proves that the additional constraint of the sub-strings exchanged to be strips makes the problem hard. It would be interesting to know what happens if we relax the constraint a bit. What if one of the sub-strings exchanged be a strip while the other could be any sub-string of the given permutation? Another interesting problem would be to find whether strip exchanging is APX-Hard.

Acknowledgments. I would like to thank my mentor Atri Rudra for being a constant source of encouragement and inspiration.

References

1. Meena Mahajan, Raghavan Rama, and S. Vijayakumar. Towards Constructing Optimal Strip Move Sequences. Lecture Notes in Computer Science, Volume 3106/2004, Pages 33-42.
2. Meena Mahajan, Raghavan Rama, Venkatesh Raman and S. Vijayakumar. Merging and Sorting By Strip Moves. Lecture Notes in Computer Science, Volume 2914/2003, Pages 314-325.
3. W.W. Bein, L.L Larmore, S. Latifi, and I.H Sudborough. Block sorting is hard. International Journal of Foundations of Computer Science, 14(3):425-437, 2003.
4. Bein W, Larmore LL, Morales L, Sudborough IH. A Faster and Simpler 2-Approximation Algorithm for Block Sorting. Proceedings of the 15th International Symposium on Fundamentals of Computation Theory, Luebeck, Germany, Lecture Notes in Computer Science 3623, Springer Verlag, 2005, pages 115-124.
5. Meena Mahajan and Raghavan Rama and S Vijayakumar. Block Sorting: A Characterization and some Heuristics. Nordic Journal of Computing, Volume 14 (2007), Pages 126-150.
6. David A. Christie. Sorting permutations by block-interchanges. Information Processing Letters. Volume 60, Issue 4, 25 November 1996, Pages 165-169
7. Lin YC, Lu CL, Chang HY, Tang CY. An efficient algorithm for sorting by block-interchanges and its application to the evolution of vibrio species. Journal of Computational Biology. Volume 12(1), 2005, Pages 102-112.
8. A. Caprara. Sorting by reversals is difficult. In Proceedings 1st Conference on Computational Molecular Biology, pages 75-83. ACM, 1997.
9. David A. Christie. A 3/2-approximation algorithm for sorting by reversals. Symposium on Discrete Algorithms. Proceedings of the ninth annual ACM-SIAM symposium on Discrete algorithms, San Francisco, California, United States. Pages: 244 - 252, 1998.
10. V. Bafna and P.A. Pevzner. Genome rearrangements and sorting by reversals. SIAM Journal on Computing, 25:272-289, 1999.
11. Piotr Berman, Sridhar Hannenhalli, and Marek Karpinski. 1.375-Approximation Algorithm for Sorting by Reversals. Lecture Notes in Computer Science, Volume 2461/2002. Pages 401-408, 2002.

12. V. Bafna and P.A. Pevzner. Sorting by transposition. SIAM Journal on Discrete Mathematics, 11:224-240, 1998.
13. Tzvika Hartman and Ron Shamir. A simpler and faster 1.5-approximation algorithm for sorting by transpositions. Information and Computation, Volume 204 , Issue 2, Pages: 275 - 290, 2006.
14. I. Elias and T. Hartman. A 1.375-Approximation Algorithm for Sorting by Transpositions IEEE/ACM Transactions on Computational Biology and Bioinformatics 2006 (TCBB), Conference version in Workshop on Algorithms in Bioinformatics 2005 (WABI'05).
15. Swapnoneel Roy and Ashok K Thakur: Approximate Strip Exchanging. International Journal of Computational Biology and Drug Design, Vol. 1, No. 1, pp.88-101, 2008.

4 Appendix

4.1 An intuition behind choosing the pivotal elements $p1$, $p2$, and $p3$, and inserting the new element e at each step

Recall that a blue edge (x, y) is drawn between adjacent elements xy such that $x > y$. Say (x, y) is the blue edge in $G(\pi)$ to be considered at the current reduction step, and y is its move element, that is the current least free element of the permutation. The intuition behind the insertion of the new element e is, we would try to create a new blue edge in $G(\pi)$ in such a way that we have a strip exchanging move to eliminate (x, y) and the newly created edge. Also we would try to do this in a way such that the move element y is one of the strips exchanged in that strip exchanging move.

Intuitively, the way out is to create a move such that y is moved after $y + 1$ to form a new strip $\{y\ y+1\}$. Again x should form a strip with the newly inserted element. Note that the strip exchanging move which exchanges y and e places e *after* x. Hence the value of e should be in between x and $x + 1$. Hence the best way seems to be inserting $e = x + 0.5$ before $y + 1$.
Now if there is nothing in front of $y + 1$, or if $y + 1$ does not have a blue edge with the element z exactly in front of it since $x > y$, we have $x + 0.5 > y + 1$. Hence $(x + 0.5, y + 1)$ forms a new blue edge. Again since $z < y$ (no blue edge between them), $z < x + 0.5$. Hence z and $x + 0.5$ do not form a blue edge. Hence in this case only one new blue edge is added to $G(\pi)$ and we are done.

If there exists a blue edge $(z, y + 1)$:

- If $x + 0.5 < z$, the blue edge $(z, y + 1)$ gets eliminated because $x + 0.5$ is inserted in between z and $y + 1$. But two new blue edges $(z, x + 0.5)$ (since $x + 0.5 < z$), and $(x + 0.5, y + 1)$ (since $x + 0.5 > y + 1$) are formed. Hence we have effectively only one new blue edge inserted here, and we are done.
- If $x + 0.5 > z$, then inserting $x + 0.5$ in between z and $y + 1$ forms only one new blue edge and disconnects the graph. The blue edge $(z, x+0.5)$ does not form. Hence to fix this, we make a different insertion here. Instead of taking element $e = x + 0.5$, we take $e = z - 0.5$ here and insert it between x and y. The idea is, we would exchange z and y in the strip exchanging move. Note that the move brings $yy + 1$ and $z - 0.5z$ together and thus two new strips get formed. But it still remains to be proved that only one new blue edge is inserted in $G(\pi)$ due to this insertion. Note that the blue edge (x, y) gets eliminated due to this insertion. Now $x + 0.5 > z$, hence $x > z - 0.5$, hence $(x, z - 0.5)$ forms a blue edge. Again since $(z, y+1)$ is a blue edge, $z > y+1$, hence $z - 0.5 > y$ and therefore $(z - 0.5, y)$ forms a new blue edge and we are done.

For the above case, the elements which determine the value and position of the element e to be inserted are x, $y + 1$, and z. These are the pivotal elements. However there are some small exceptions to this. Element x might have been

inserted in a previous step by the reduction. In that case, we consider the element w which is the first element in the current not inserted in any previous step. The second step in Figure 2 is such an example. In this step, $(3.5, 2)$ is the blue edge currently under consideration. But since 3.5 was inserted in a previous step, we consider the pivotal element as 5 instead of it.

4.2 An example of execution of the reduction procedure

We illustrate the reduction procedure with an example permutation π = 4 3 1 9 5 2 11 8 6 10 7 12 in Figure 1. π' = 4 3 1 9 5.5 5 3.7 3.5 2 11 8 6 3.6 10 8.5 7 12 is the output of the reduction. Each move element i its $move(i)$ shown in the boxes in the figure. All the blue edges are marked at the end of the reduction. Note that this is a *YES* instance of the problem.

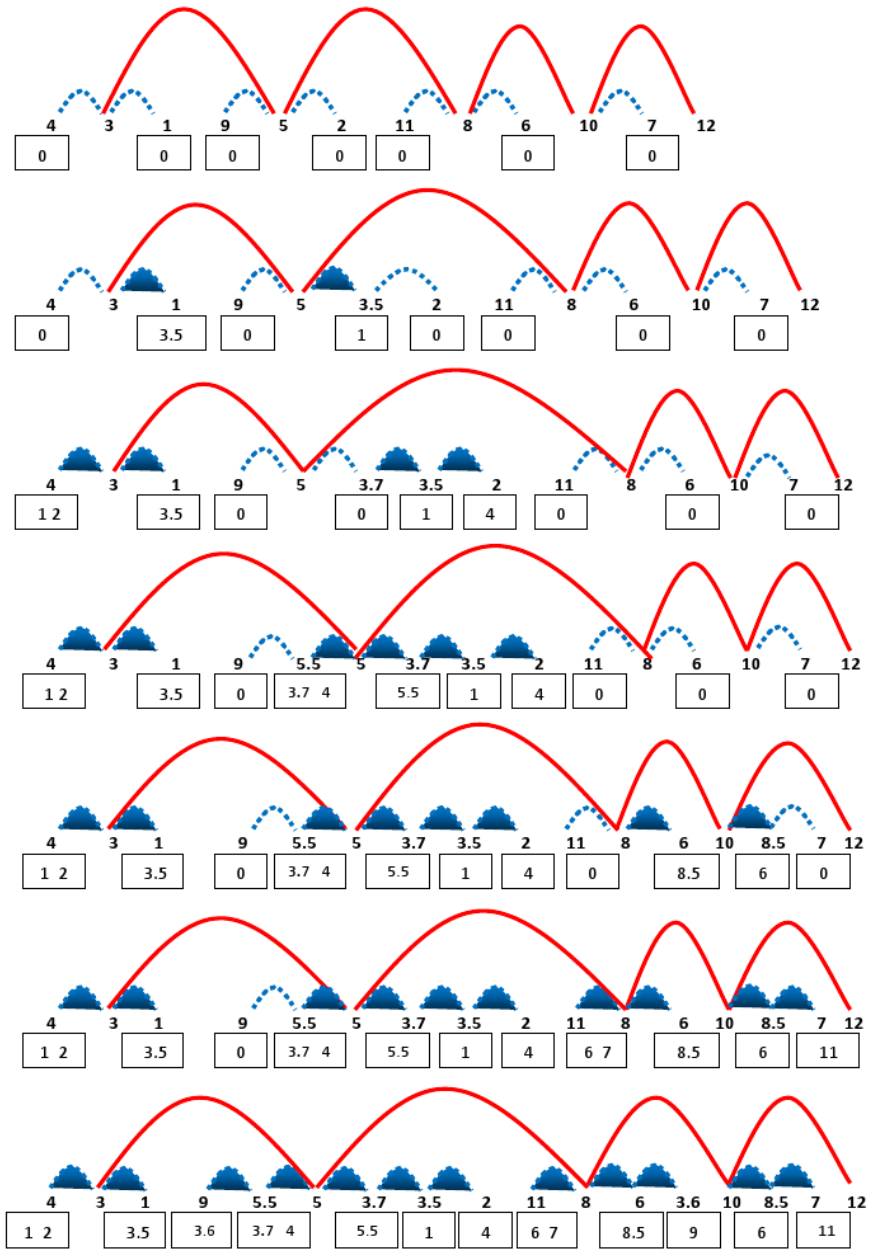

Fig. 1. Execution of the reduction procedure on a permutation.

Finally we include the subroutines findPivotalMore and placeAtBack, which are almost alike the findPivotalLess and placeInFront subroutines. We had omitted these subroutines earlier due to space constraints:

Input: The elements x and y
Output: The pivotal elements $p1$, $p2$, and $p3$
`/* This subroutine is called when` x `is least free element */`
`/* Decide` $p1$ `*/`
if *move*(y) *does not exist, or move*(y) *is equal to* 0 **then**
  $p1 = y$;
end
else
  `/* This case arises if` y `has been inserted at a previous step of the reduction. */`
  Start from y and traverse the chain containing x and y towards right;
  Stop at element w, where w is either not a move element, or not a free element;
  $p1 = w$;
end
`/* Decide` $p2$ `*/`
if x *is the leftmost element of the chain containing* x **then**
  $p2 = prev(x)$
end
else
  Let l be the leftmost element of the chain containing x;
  if *move*(r) *is equal to* $prev(x)$ **then**
    $p2 = prev(prev(x))$;
  end
  else
    $p2 = prev(x)$;
  end
end
`/* Decide` $p3$ `*/`
if z *is just after* $p2$ *and there exists a blue edge between* z *and* $p2$ **then**
  $p3 = z$;
end
else
  $p3 = 0$;
end

Algorithm 4: The findPivotalMore(x, y) subroutine

Input: The elements x and y and the pivotal elements $p1$, $p2$, and $p3$
Output: A new element e to be inserted and its position to be inserted based on some cases
if $p3$ *is equal to* 0 **then**
  e is an element with a value between $p1$ and $prev(p1)$ i.e $p1 > e > prev(p1)$;
  Insert e just after $p2$ (adjacent to $p2$);
  Set $move(x)$ to e and $move(e)$ to x ;
  Mark blue edges (x, y) and $(e, p2)$;
end
else
  `/* p3 is not equal to 0 */`
  if $p3$ *is free and is equal to* $prev(p1)$ **then**
    `/* This is the only case in which we do not need to insert any new element */`
    Set $move(x)$ to $p3$ and $move(p3)$ to x;
    Mark blue edges (x, y) and $(p3, p2)$;
  end
  if *(*$p3$ *is less than* $prev(p1)$*) or (*$p3$ *is not free and is equal to* $prev(p1)$*)* **then**
    e is an element with a value between $p1$ and $next(p1)$ i.e $p1 > e > prev(p1)$;
    Insert e between $p3$ and $p2$;
    Set $move(x)$ to e and $move(e)$ to x;
    Mark blue edges (x, y) and $(e, p2)$;
  end
  if $p3$ *is greater than* $p1$ **then**
    e is an element with a value between $next(p3)$ and $p3$ i.e $p3 < e < next(p3)$;
    Insert e right before p1 i.e at position $\pi(p1) - 1$;
    Set $move(x)$ to $p3$ and $move(p3)$ to x;
    if $p1$ *is not equal to* y **then**
      Mark blue edges (x, y) and $(p3, p2)$;
    end
    else
      Mark blue edges (x, e) and $(p3, p2)$;
    end
  end
end

Algorithm 5: The placeAtBack(x, y, $p1$, $p2$, $p3$) subroutine

For a recap, these two subroutines are called from the main reduction procedure if and when x (the greater element) is the move element for the current blue edge (x, y) under consideration.

4.3 An example of the application of strip exchanging in OCR

In figure 2, we illustrate this concept with an example inspired by an application in optical character recognition. Here we have a permutation "How ? they did it do" recognized, but not in the correct order "How they did do it ?". We observe that it requires 2 strip exchanging moves to sort the permutation. The strips have been moved and combined with other strips to form larger strips at each step.

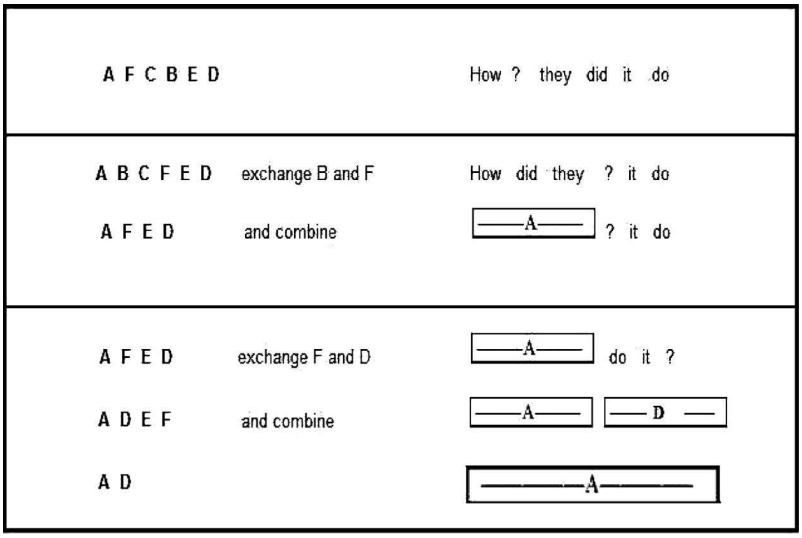

Fig. 2. How to sort "How ? they did it do" using strip exchanging.