SIAM J. DISCRETE MATH. © 2001 Society for Industrial and Applied Mathematics
Vol. 14, No. 2, pp. 193–206

SORTING STRINGS BY REVERSALS AND BY TRANSPOSITIONS*

DAVID A. CHRISTIE† AND ROBERT W. IRVING†

Abstract. The problems of sorting by reversals and sorting by transpositions have been studied because of their applications to genome comparison. Prior studies of both problems have assumed that the sequences to be compared (or sorted) contain no duplicates, but there is a natural generalization in which the sequences are allowed to contain repeated characters. In this paper we study primarily the versions of these problems in which the strings to be compared are drawn from a binary alphabet. We obtain upper and lower bounds for reversal and transposition distance and show that the problem of finding reversal distance between binary strings, and therefore between strings over an arbitrary fixed-size alphabet, is NP-hard.

Key words. strings, sorting, genome comparison, reversals, transpositions, NP-complete problems

AMS subject classifications. 68R05, 68R15, 68Q17, 92D15

PII. S0895480197331995

1. Introduction.

1.1. Reversals and transpositions over permutations.

The reversal distance and the transposition distance between two permutations (and the related problems of sorting by reversals and sorting by transpositions) are used to estimate the number of global mutations between genomes and can be used by molecular biologists to infer evolutionary and functional relationships between genomes. A *reversal* (or inversion) involves reversing the order of elements in a substring of the permutation. More formally, the reversal $\rho(i,j)$ $(1 \le i < j \le n)$ transforms the permutation π of $\{1,2,\ldots,n\}$ into π', where

$$\pi'(k) = \begin{cases} \pi(i+j-k) & \text{if } i \le k \le j, \\ \pi(k) & \text{otherwise.} \end{cases}$$

A *transposition* involves swapping two adjacent substrings of the permutation. More formally, the transposition $\tau(i,j,k)$ $(1 \le i < j < k \le n+1)$ transforms the permutation π of $\{1,2,\ldots,n\}$ into π', where

$$\pi'(m) = \begin{cases} \pi(m+j-i) & \text{if } i \le m < i+k-j, \\ \pi(m-k+j) & \text{if } i+k-j \le m < k, \\ \pi(m) & \text{otherwise.} \end{cases}$$

Sorting by reversals is the problem of finding the minimum number $d_r(\pi)$ of reversals needed to transform a given permutation π into the identity permutation $\imath$. *Sorting by transpositions* is the analogous problem of determining $d_t(\pi)$, the minimum number of transpositions needed to transform π into $\imath$. The functions $d_r(\pi)$ and $d_t(\pi)$ are known as the *reversal distance* and *transposition distance*, respectively, of π.

In the context of sorting by reversals, Kececioglu and Sankoff [14] introduced the concept of a breakpoint. A permutation π of $\{1,2,\ldots,n\}$ has a *breakpoint* at position

*Received by the editors December 31, 1997; accepted for publication (in revised form) December 21, 2000; published electronically March 15, 2001.
http://www.siam.org/journals/sidma/14-2/33199.html

†Department of Computing Science, University of Glasgow, Glasgow G12 8RZ, Scotland, UK (christie@dcs.gla.ac.uk, rwi@dcs.gla.ac.uk).

i if $|\pi(i) - \pi(i-1)| \neq 1$. (Special elements $\pi(0) = 0$ and $\pi(n+1) = n+1$ are added so that breakpoints at the ends of the permutation are included.) Consideration of breakpoints leads to simple lower and upper bounds for reversal distance [14]. The *reversal diameter* of the symmetric group S_n is the maximum value of $d_r(\pi)$ over all permutations of length n. Bafna and Pevzner [1] have proved that the reversal diameter of S_n is $n-1$ and is achieved by only two permutations of length n.

In the transposition case, Bafna and Pevzner [2] defined breakpoints slightly differently. Here, π has a breakpoint at position i if $\pi(i) - \pi(i-1) \neq 1$, and once again, consideration of breakpoints leads to simple lower and upper bounds for transposition distance [2]. The *transposition diameter* of S_n is the maximum value of $d_t(\pi)$ over all permutations of length n. The transposition diameter of S_n has not been resolved, but Bafna and Pevzner [2] have shown that it lies somewhere between $n/2+1$ and $3n/4$.

Caprara [4] (see also [5]) has shown that sorting by reversals is NP-hard. Earlier, Kececioglu and Sankoff [14] had found a simple 2-approximation algorithm and Bafna and Pevzner [1] a 7/4-approximation algorithm. Christie [7] has obtained a 3/2-approximation algorithm, which is the best currently known for this problem.

There is a variation of sorting by reversals in which each element of the permutation is given a sign "+" or "−" that is flipped when the element is involved in a reversal. Perhaps surprisingly, this version of the problem is solvable in polynomial time, as was proved by Hannenhalli and Pevzner [11] (see also [12]). Their algorithm to find signed reversal distance has been improved and simplified by Berman and Hannenhalli [3] and also by Kaplan, Shamir, and Tarjan [13].

Sorting by transpositions is less well understood than sorting by reversals, and in particular, the complexity of sorting by transpositions remains open. However, Bafna and Pevzner [2] have described a 3/2-approximation algorithm for the problem.

1.2. Reversals and transpositions over strings. In the context of genome comparisons, duplicate genes can occur, so that the permutation model is not always the appropriate one. In this paper, we define reversal distance and transposition distance on strings and investigate these new problems, which are of interest in their own right, focusing primarily on the case of a binary alphabet. However, some of the bounds that we establish can be extended to arbitrary fixed-size alphabets, and our main NP-hardness result—for reversal distance over a binary alphabet—immediately implies NP-hardness of the corresponding problem over an arbitrary fixed-size alphabet.

For permutations, transforming π into ρ is equivalent to transforming $\rho^{-1} \cdot \pi$ into $\imath$. However, there is no direct analogue of this result for strings. Therefore in this context the problems are expressed in terms of the distance between two strings.

For strings S and T, the *reversal distance* $d_r(S,T)$ between S and T is the minimum number of reversals required to transform S into T; and the *transposition distance* $d_t(S,T)$ is the minimum number of transpositions required to transform S into T. It is impossible to insert or delete characters using reversals or transpositions, so in each case T must be a rearrangement of S. We say that S and T are *related* if T is a rearrangement of S.

Note that the sorting problem on permutations is a special case of the distance problem on strings. Therefore, for reversals and transpositions, the distance problem on strings is at least as hard as the sorting problem on permutations. Thus, finding the reversal distance between strings is NP-hard since sorting permutations by reversals is NP-hard. However, if the strings are drawn from a fixed-size alphabet, then minimally sorting a permutation is no longer a special case of finding the distance between two strings, and the hardness of sorting permutations by reversals does not imply a corresponding result for sorting strings by reversals in this case.

The remainder of the paper is organized as follows. Section 2 introduces appropriate terminology and notation. Section 3 is devoted to reversals and section 4 to transpositions. In these two sections lower bounds, upper bounds, and diameter results are described for each problem, respectively. In section 5 it is proved that the generalized version of sorting by reversals is NP-hard, even when the strings are drawn from a binary alphabet. In the final section we compare and contrast the results obtained for binary strings with the known results for sorting problems over permutations.

2. Terminology and notation.

We denote the ith symbol of a string S by $S(i)$. A reversal on a string will be represented by enclosing in brackets the substring to be reversed. For example, 0[1010110]1 = 001101011. A similar notation, in which two adjacent substrings are bracketed, will be used to describe transpositions—for instance, 0[10][1011]01 = 010111001.

Let 0^k represent a string of zeros of length k, 1^k a string of ones of length k, and, in general, let S^k (or $(S)^k$) represent the string obtained by concatenating k copies of S for any string S. We use $S \cdot T$, or simply ST, to denote the concatenation of strings S and T, and we define a string concatenation operation ($\sum$) that can be used in a similar way to summation. For example, $\sum_{i=1}^{3}(0^i 1) = 010010001$. We also use the standard notation S^+ to represent the concatenation of one or more copies of S and S^* the concatenation of zero or more copies of S.

Let $\mathcal{B}_n$ be the set of binary strings of length n. For a particular transformation, the *diameter* of $\mathcal{B}_n$ is the maximum distance between any two related binary strings of length n. Define $E_k = 0^k 1^k$ and $C_k = (10)^k$. For example, $E_4 = 00001111$ and $C_4 = 10101010$. These strings are particularly useful for establishing diameter results in later sections.

Let $\overline{S}$ represent the string derived from S by switching ones and zeros, and let S^R represent the string S in reverse order. Therefore, for example, if $S = 0100110001$, then $\overline{S} = 1011001110$, $S^R = 1000110010$, and $\overline{S}^R = 0111001101$.

Strings X and Y are *isomorphic* to strings S and T if

(i) $X = S$ and $Y = T$, or $X = T$ and $Y = S$, or
(ii) $X = \overline{S}$ and $Y = \overline{T}$, or $X = \overline{T}$ and $Y = \overline{S}$, or
(iii) $X = S^R$ and $Y = T^R$, or $X = T^R$ and $Y = S^R$, or
(iv) $X = \overline{S}^R$ and $Y = \overline{T}^R$, or $X = \overline{T}^R$ and $Y = \overline{S}^R$.

In other words, the pairs $\{X, Y\}$ and $\{S, T\}$ are isomorphic if one pair can be obtained from the other by a fixed permutation of the alphabet, followed by an optional complete reversal of both strings in the pair. This version of the definition applies equally to strings over an arbitrary alphabet. Obviously, if X and Y are isomorphic to S and T, then $d_r(X, Y) = d_r(S, T)$ and $d_t(X, Y) = d_t(S, T)$.

Define $lcp(S, T)$ and $lcs(S, T)$ to be the lengths of the longest common prefix and the longest common suffix, respectively, of S and T.

A *block of zeros* is a maximal length substring that consists only of the character 0. A *block of ones* is defined similarly. Let $b(S)$ denote the total number of blocks in S and $z(S)$ denote the number of blocks of zeros in S. Therefore, for example, $b(001110101) = 6$ and $z(001110101) = 3$.

Finally, we represent by $\ldots$ an arbitrary substring of length ≥ 0. For example, if S has prefix "01," a substring "00," and suffix "11," then we could write $S = 01\ldots00\ldots11$.

3. Reversal distance between binary strings.

In this section, we describe a lower bound and an upper bound for reversal distance between binary strings. These bounds are then used to determine the reversal diameter of $\mathcal{B}_n$ and also to identify some strings that achieve this reversal diameter. A restricted version of the problem, that is in some sense analogous to sorting permutations by reversals, is shown to be solvable in polynomial time. However, in section 5 the general problem of determining reversal distance between two (binary) strings is shown to be NP-hard.

3.1. A lower bound.

We first adapt the concept of breakpoint from permutation sorting problems for use in the context of string sorting. This new kind of breakpoint is then used to establish a lower bound for reversal distance. Recall that, for permutations, two elements form a breakpoint if they are adjacent in π but not adjacent in the identity permutation. Substrings of length two represent adjacencies in strings S and T, so our definition of breakpoints on strings will be based on these substrings.

If S contains more "00" substrings than T, then each extra "00" must be broken, by a reversal, at some time in the transformation from S into T. Each extra "00" in S is an example of a *reversal breakpoint*. An obvious difference between breakpoints on strings and on permutations is that, on strings, the specific location of a breakpoint may not necessarily be identified. For instance, if S contains three "00" substrings and T contains only two "00" substrings, then one of the "00" substrings in S is a breakpoint, but no particular "00" substring of S is identified as the breakpoint. Breakpoints also occur for "01," "10," and "11" substrings as well. However, because reversals can convert "01" substrings into "10" substrings and vice versa, these substrings must be counted together when considering reversal breakpoints.

Breakpoints can also be contributed from the beginning and end of the strings. For example, if $S(1) \neq T(1)$, then position one contributes a breakpoint. In order to deal with these breakpoints, S and T are extended by adding special characters α at the beginning and ω at the end of both strings. These breakpoints can then be counted by comparing the number of occurrences of the substrings "$\alpha0$," "$\alpha1$," "0ω," and "1ω" in both strings. Adding α and ω to S and T is similar to adding 0 and $n+1$ to π when dealing with permutations.

The number of times the substring "ab" occurs in S, i.e., the frequency count for "ab" in S, is denoted by $f_{ab}(S)$, where $a, b \in \{\alpha, 0, 1, \omega\}$. We also assume, for convenience, $\alpha < 0 < 1 < \omega$.

We now define the number of reversal breakpoints between S and T, $b_r(S,T)$ to be

$$b_r(S,T) = \sum_{\alpha \leq a < b \leq \omega} \delta(f_{ab}(S) + f_{ba}(S) - f_{ab}(T) - f_{ba}(T)) + \sum_{0 \leq a \leq 1} \delta(f_{aa}(S) - f_{aa}(T)),$$

where

$$\delta(x) = x \text{ if } x > 0 \text{ and } 0 \text{ otherwise.}$$

Clearly, if $S = T$, then $b_r(S,T) = 0$. However, it is possible to have $b_r(S,T) = 0$, even when $S \neq T$, for example, if $S = 100101$ and $T = 101001$.

Note that, although the definition of the number of breakpoints is not symmetric with respect to the two strings S and T, it is easy to see that $b_r(S,T) = b_r(T,S)$.

We now derive a lower bound for reversal distance based on these breakpoints.

LEMMA 3.1. *Suppose that S' is obtained from S by a single reversal. Then*

$$b_r(S',T) \geq b_r(S,T) - 2.$$

Proof. A reversal on S cuts two substrings of length two in the extended version of S. Hence the number of breakpoints can be reduced by at most two as a result of such a reversal, and the result follows. ∎

This lemma can be used to easily deduce the following lower bound for reversal distance.

THEOREM 3.2. *Let S and T be related binary strings. Then*

$$d_r(S,T) \geq \lceil b_r(S,T)/2 \rceil.$$

It is easy to find examples for which this lower bound is not tight; e.g., if $S = 0011000111$ and $T = 1110011000$, then $b_r(S,T) = 2$ and $d_r(S,T) = 2 \neq \lceil b_r(S,T)/2 \rceil$.

The definition of a reversal breakpoint for binary strings, together with the bound of Theorem 3.2, can be generalized in a straightforward way to strings over larger alphabets.

3.2. An upper bound. In this section we derive a simple upper bound on reversal distance for binary strings.

LEMMA 3.3. *Let S and T be related strings of length n such that $S \neq T$. Then it is possible either*

(a) *to apply a reversal on S resulting in the string S', such that $lcp(S',T) + lcs(S',T) \geq lcp(S,T) + lcs(S,T) + 2$, or*

(b) *to apply a reversal on T resulting in the string T' such that $lcp(S,T') + lcs(S,T') \geq lcp(S,T) + lcs(S,T) + 2$.*

Proof. Without loss of generality, it can be assumed that $S(1) = 0$, since otherwise we could consider $\overline{S}$ and $\overline{T}$ and apply the resulting reversal to S or T. Further, it can be assumed that $S(1) \neq T(1)$, and $S(n) \neq T(n)$, because otherwise we could reduce n by removing any common prefix or suffix from both strings.

The reversal applied to S or T depends on S and T. We describe seven cases and show a reversal with the required property in each case. (Each case is considered only if S and T fail to meet the conditions of any of the earlier cases, and the cases are thereby mutually exclusive.)

Case (i). $S(n) = 1$: then $S = 0\ldots1$ and $T = 1\ldots0$, so take $S' = [0\ldots1]$.

Case (ii). $T(2) = 0$: then $S = 0\ldots0$ and $T = 10\ldots1$, so take $S' = [0^+1]\ldots0$.

Case (iii). $T(n-1) = 0$: then $S = 0\ldots0$ and $T = 11\ldots01$, so, by considering S^R and T^R, this case can be dealt with in a similar way to Case (ii).

Case (iv). $f_{11}(S) > 0$: then $S = 0\ldots11\ldots0$ and $T = 11\ldots11$, so take $S' = [0^+(10^+)^*11]\ldots0$.

Case (v). $S(2) = 1$: then $S = 01\ldots0$ and $T = 11\ldots11$, so, by considering $\overline{T}$ and $\overline{S}$, this case can be dealt with in a similar way to Case (ii).

Case (vi). $S(n-1) = 1$: then $S = 00\ldots10$ and $T = 11\ldots11$, so, by considering $\overline{T}^R$ and $\overline{S}^R$, this case can be dealt with in a similar way to Case (ii).

Case (vii). $f_{00}(T) > 0$: then $S = 00\ldots00$ and $T = 11\ldots00\ldots11$, so, by considering $\overline{T}$ and $\overline{S}$, this case can be dealt with in a similar way to Case (iv).

This completes the proof because Cases (i)–(iv) can fail to apply only if S contains more zeros than ones, whereas Cases (i) and (v)–(vii) can fail to apply only if T contains more ones than zeros. ∎

THEOREM 3.4. *Let S and T be related binary strings of length n. Then*

$$d_r(S,T) \leq \lfloor n/2 \rfloor.$$

Proof. Lemma 3.3 describes a way to increase the combined length of the common prefix and suffix of S and T by at least two using a single reversal. Therefore a sequence of $\lfloor n/2 \rfloor$ such reversals will be enough to transform S into T. ☐

For example, if $S = 010101010$ and $T = 110000011$, then applying reversals as described in the proof of Lemma 3.3 results in the following sequence of strings:

$$\begin{array}{cccccccccc}
0 & 1 & [\,0 & 1\,] & 0 & 1 & 0 & 1 & 0 \\
0 & 1 & 1 & 0 & 0 & [\,1 & 0 & 1 & 0\,] \\
0 & 1 & 1 & 0 & 0 & 0 & [\,1 & 0\,] & 1 \\
[\,0 & 1 & 1\,] & 0 & 0 & 0 & 0 & 1 & 1 \\
1 & 1 & 0 & 0 & 0 & 0 & 0 & 1 & 1
\end{array}$$

Note that the first reversal found by the proof of Lemma 3.3 is the one that reverses the first three symbols of T, and so it is the last reversal shown in the illustration.

3.3. Reversal diameter of $\boldsymbol{\mathcal{B}_n}$. The *reversal diameter*, $D_r(n)$, of $\mathcal{B}_n$ is defined to be the maximum value of $d_r(S,T)$ over all related binary strings S and T of length n. More formally

$$D_r(n) = \max\{d_r(S,T) : S,\ T \text{ are related binary strings of length } n\}.$$

LEMMA 3.5. *$\forall k \geq 1$, $d_r(E_k, C_k) = k$ and $d_r(0 \cdot E_k, 0 \cdot C_k) = k$.*

Proof. This follows at once by application of Theorems 3.2 and 3.4 to these strings. ☐

THEOREM 3.6. *$\forall n \geq 1$, $D_r(n) = \lfloor n/2 \rfloor$.*

Proof. This is an immediate consequence of Theorem 3.4 and Lemma 3.5. ☐

THEOREM 3.7. *Let S and T be related binary strings of length $2n \geq 6$. Then $d_r(S,T) = n$ if and only if S and T are isomorphic to C_n and E_n.*

Proof. We prove this theorem by induction. The base case is when $n = 3$. Then, by complete search, it may be verified that $d_r(S,T) = 3$ if and only if S and T are isomorphic to E_3 and C_3. Now suppose that the theorem holds when $n \leq k$. Let S and T be strings of length $2k+2$ such that $d_r(S,T) = k+1$. We show that S and T are isomorphic to C_{k+1} and E_{k+1}.

By Lemma 3.3, we can apply a reversal to S or T that increases the combined length of the common prefix and suffix by at least two. Without loss of generality, we can relabel S and T so that the reversal found in the proof of Lemma 3.3 is applied to S resulting in the string S'. Furthermore, we can assume, without loss of generality, that $S(1) = 0$.

It must be that $lcp(S',T) + lcs(S',T) = 2$ and $d_r(S',T) = k$, since any alternative would contradict $d_r(S,T) = k+1$. Let S'_e and T_e be the strings S' and T excluding any common prefix and suffix. By the induction hypothesis, S'_e and T_e must be isomorphic to E_k and C_k. Therefore, since $\overline{E_k} = {E_k}^R$ and $\overline{C_k} = {C_k}^R$, either (a) $S'_e = E_k$ and $T_e = C_k$, or (b) $S'_e = C_k$ and $T_e = E_k$, or (c) $S'_e = {E_k}^R$ and $T_e = {C_k}^R$, or (d) $S'_e = {C_k}^R$ and $T_e = {E_k}^R$.

By the proof of Lemma 3.3 there are essentially three ways that the reversal can be applied to S, as typified by Cases (i), (ii), and (iv) in that proof. We take each of these three cases in turn and show that for the four possible values of S'_e and T_e, $d_r(S,T) = k+1$ if and only if S and T are isomorphic to E_{k+1} and C_{k+1}.

Case (i). $S = 0\ldots1$, $T = 1\ldots0$. In this case the whole of S is reversed. We show that cases (a) and (b) for S'_e and T_e lead to contradictions, whereas cases (c) and (d) establish the induction step.

(a) $S = 0\cdot{E_k}^R\cdot1 = 0\cdot1^k\cdot0^k\cdot1$ and $T = 1\cdot C_k\cdot0 = 1\cdot(10)^k\cdot0$. Suppose that instead of applying the reversal of Lemma 3.3 we apply the reversal $[011]1^{k-2}\cdot0^k\cdot1$ to obtain S''. This reversal extends the common prefix by three characters, so $d_r(S'',T) < k$. Therefore $d_r(S,T) < k+1$, a contradiction.

(b) $S = 0\cdot{C_k}^R\cdot1 = 0\cdot(01)^k\cdot1$ and $T = 1\cdot E_k\cdot0 = 1\cdot0^k\cdot1^k\cdot0$. These strings are isomorphic to the strings in (a), so $d_r(S,T) < k+1$, a contradiction.

(c) $S = 0\cdot E_k\cdot1 = E_{k+1}$ and $T = 1\cdot{C_k}^R\cdot0 = C_{k+1}$.

(d) $S = 0\cdot C_k\cdot1 = {C_{k+1}}^R$ and $T = 1\cdot{E_k}^R\cdot0 = {E_{k+1}}^R$.

Case (ii). $S = 0\ldots0$ and $T = 10\ldots1$. In this case the reversal results in a string S' that has prefix "10." Therefore S'_e and T_e must be suffixes of S' and T. Now since T ends with a 1, only cases (b) and (c) need to be considered. Both cases lead to a contradiction.

(b) $S' = 10\cdot C_k = 10\cdot(10)^k$ and $T = 10\cdot E_k = 10\cdot0^k\cdot1^k$. The reversal on S ends with the first "1" in S, so $S = 01\cdot(10)^k$. Then the reversal $[01101]0\cdot(10)^{k-2}$ produces string S'' that has $d_r(S'',T) < k$ by the induction hypothesis. Therefore $d_r(S,T) < k+1$, a contradiction.

(c) $S' = 10\cdot{E_k}^R = 10\cdot1^k\cdot0^k$ and $T = 10\cdot{C_k}^R = 10\cdot(01)^k$. Therefore $S = 01\cdot1^k\cdot0^k$, because the reversal on S ends with the first "1." Then the reversal $01\cdot1^{k-1}[10^k]$ results in a string S'' that has $d_r(S'',T) < k$ by the induction hypothesis. Therefore $d_r(S,T) < k+1$, a contradiction.

Case (iv). $S = 0\ldots11\ldots0$ and $T = 11\ldots11$. In this case the reversal is applied to S to obtain a string S' that has prefix 11. Therefore S'_e and T_e must be suffixes of S' and T. Now, since T ends with "11" only case (b) need be considered. However, in fact, even this case cannot occur.

(b) $S' = 11\cdot C_k = 11\cdot(10)^k$ and $T = 11\cdot E_k = 11\cdot0^k\cdot1^k$. However, then the reversal on S could not have moved the first "11" substring in S. Therefore this case cannot occur.

Therefore $d_r(S,T) = k+1$ if and only if S and T are isomorphic to E_{k+1} and C_{k+1}. Therefore, by induction, we have proved the theorem. $\square$

Theorem 3.7 describes the strings of length n that achieve the reversal diameter when n is even. When n is odd, significantly more pairs of strings achieve the reversal diameter.

We note in passing that there appears to be no simple analogue of Lemma 3.3 in the case of alphabet size > 2 and therefore no easy generalization of Theorem 3.6.

3.4. Sorting by reversals. Let $S_\imath$ denote the string that is related to S and consists only of a block of zeros, followed by a block of ones. For example, if $S = 01100110$, then $S_\imath = 00001111$. Then determining $d_r(S,S_\imath)$ is an analogue of determining the reversal distance of a permutation. We show that $d_r(S,S_\imath)$ can be determined in polynomial time.

Recall that $z(S)$ denotes the number of blocks of zeros contained in S. Obviously, $z(S_\imath) = 1$ (unless S does not contain any zeros). The following lemma can be verified easily.

LEMMA 3.8. *Let S' be a string obtained from S by a single reversal. Then*

$$z(S') \geq z(S) - 1.$$

With this lemma we can determine $d_r(S, S_\imath)$.

THEOREM 3.9. *For any binary string S,*

$$d_r(S, S_\imath) = \begin{cases} z(S) - 1 & \textit{if } S(1) = 0, \\ z(S) & \textit{otherwise.} \end{cases}$$

Proof. By Lemma 3.8, $d_r(S, S_\imath) \geq z(S) - 1$. If $S(1) = 0$, then $z(S) - 1$ reversals of the form $0^+[1^+0^+]1\ldots$ or $0^+[1^+0^+]$ transform S into $S_\imath$. If $S(1) = 1$, then an extra reversal is required because it is impossible to change the first symbol to a 0 and also reduce the value of z. This bound can be achieved by performing a reversal $[1\ldots0]1\ldots$ or $[1\ldots0]$ before performing $z(S) - 1$ reversals as described for the case $S(1) = 0$. ◻

The distance described in Theorem 3.9 can be calculated easily in polynomial time. In section 5 it is shown that, in general, determining $d_r(S, T)$ is NP-hard.

4. Transposition distance between binary strings.

In this section, we present an upper bound and a lower bound for transposition distance between binary strings. These bounds are used to determine the transposition diameter, and identify some strings that achieve the diameter. A restricted version of the problem, that is analogous to the problem of sorting by transpositions, is shown to be solvable in polynomial time.

4.1. A lower bound.

Again, it is the appropriate concept of a breakpoint that is used to obtain a lower bound for transposition distance.

Transposition breakpoints are defined in a similar way to reversal breakpoints. For example, if S contains more "11" substrings than T, then each extra "11" substring contributes a breakpoint. However, a crucial difference between reversal breakpoints and transposition breakpoints is that "01" and "10" substrings are counted separately, since a transposition cannot transform one to the other. As before, we prepend α and append ω to each string.

The number of *transposition breakpoints* is therefore

$$b_t(S, T) = \sum_{a,b \in A} \delta(f_{ab}(S) - f_{ab}(T)),$$

where $A = \{\alpha, 0, 1, \omega\}$ and, as before,

$$\delta(x) = x \text{ if } x > 0 \text{ and } 0 \text{ otherwise.}$$

Clearly, if $S = T$, then $b_t(S, T) = 0$. However, it is possible that $b_t(S, T) = 0$, even when $S \neq T$, for example, if $S = 101001$ and $T = 100101$.

LEMMA 4.1. *Suppose that S' is obtained from S by a single transposition. Then*

$$b_t(S', T) \geq \begin{cases} b_t(S, T) - 3 & \textit{if } S(1) \neq S'(1) \textit{ and } S(n) \neq S'(n), \\ b_t(S, T) - 2 & \textit{otherwise.} \end{cases}$$

Proof. The transposition must have the form $\ldots a[b\ldots c][d\ldots e]f\ldots$, where $a \in \{\alpha, 0, 1\}$, $b, c, d, e \in \{0, 1\}$, and $f \in \{0, 1, \omega\}$. The transposition results in the string $\ldots a[d\ldots e][b\ldots c]f\ldots$. Now let us suppose that the none of the substrings "ab," "cd," or "ef" is the same as any of the substrings "ad," "eb," or "cf." Then "cd"$\neq$"cf," so $d \neq f$. Similarly, $c \neq a$, $b \neq f$, and $e \neq a$. Now suppose that $a \neq \alpha$. Then $c = e$. However, then "ef"="cf", a contradiction. Similarly if $f \neq \omega$, then "ab"="ad." Therefore, if $a \neq \alpha$ or $f \neq \omega$, then at least one of the substrings "ab," "cd," or "ef" is the same as one of the substrings "ad," "eb," or "cf."

If a transposition moves the first and last symbol of S, then at most three substrings of length two may change as a result of the transposition. Given the definition of $b_t(S', T)$, this means that $b_t(S', T) \geq b_t(S, T) - 3$. However, if a transposition does not move the first and last symbols of S, then at most two substrings of length two may change as a result of the transposition. In such cases $b_t(S', T) \geq b_t(S, T) -2$. □

THEOREM 4.2. *Let S and T be related binary strings of length n. Then*

$$d_t(S,T) \geq \begin{cases} \lceil b_t(S,T)/2 \rceil & \text{if } S(1) = T(1), \text{ or } S(n) = T(n), \\ \lceil (b_t(S,T) - 1)/2 \rceil & \text{otherwise.} \end{cases}$$

Proof. A sequence of transpositions that transforms S into T can contain at most one transposition that reduces the number of breakpoints by 3. Such a transposition is only possible if $S(1) \neq T(1)$ and $S(n) \neq T(n)$. Every other transposition can reduce the number of breakpoints by at most 2. The theorem follows easily from these observations. □

As before, it is easy to find examples for which this lower bound is not exact; for example, if $S = 011100110001$ and $T = 100011001110$, then the bound is 1, but $d_t(S, T) = 2$.

Again, the definition of a transposition breakpoint for binary strings together with the bound of Theorem 4.2 can be directly extended to strings over larger alphabets. For alphabets of size > 2, however, each transposition can reduce the number of breakpoints by 3. Hence the extended version of Theorem 4.2 is as follows.

THEOREM 4.3. *Let S and T be related strings, of length n, over an alphabet of size > 2. Then*

$$d_t(S,T) \geq \lceil b_t(S,T)/3 \rceil.$$

4.2. An upper bound.

In this section a simple upper bound is derived for transposition distance.

LEMMA 4.4. *Let S and T be related strings of length n such that $S \neq T$. Then it is possible either*

(a) *to apply a transposition to S resulting in a string S' such that $lcp(S', T) + lcs(S', T) \geq lcp(S, T) + lcs(S, T) + 2$ or*

(b) *to apply a transposition to T resulting in a string T' such that $lcp(S, T') + lcs(S, T') \geq lcp(S, T) + lcs(S, T) + 2$.*

Proof. Without loss of generality, it can be assumed that $S(1) = 0$, $S(1) \neq T(1)$, and $S(n) \neq T(n)$. The transposition that is applied to S or T depends on S and T. Three cases are described, and for each case a transposition is shown with the required property. (Again, each case is considered only if S and T fail to meet the conditions of any earlier case, making the cases mutually exclusive.)

Case (i). $S(n) = 1$: then $S = 0 \ldots 1$ and $T = 1 \ldots 0$, so take $S' = [0^+][1 \ldots]$.

Case (ii). $f_{11}(S) > 0$: then $S = 0 \ldots 11 \ldots 0$ and $T = 1 \ldots 1$, so take $S' = [0^+(10^+)^*1][1 \ldots 0]$.

Case (iii). $f_{11}(S) = 0$ and $f_{00}(T) > 0$: then $S = 0 \ldots 0$ and $T = 1 \ldots 00 \ldots 1$, so, by considering $\overline{T}$ and $\overline{S}$, this case is similar to Case (ii).

This completes the proof because Cases (i) and (ii) can fail to apply only if S contains more zeros than ones, whereas Cases (i) and (iii) can fail to apply only if T contains more ones than zeros. ◻

THEOREM 4.5. *Let S and T be related binary strings of length n. Then*

$$d_t(S,T) \le \lfloor n/2 \rfloor.$$

Proof. Lemma 4.4 describes a way to increase the combined length of the common prefix and suffix of S and T by at least two using a single transposition. Therefore a sequence of $\lfloor n/2 \rfloor$ such transpositions will be enough to transform S into T. ◻

4.3. Transposition diameter of $\mathcal{B}_n$. The *transposition diameter*, $D_t(n)$, of $\mathcal{B}_n$ is the maximum value of $d_t(S,T)$ taken over all related binary strings of length n. More formally

$$D_t(n) = \max\{d_t(S,T) : \text{S, T are related binary strings of length } n\}.$$

LEMMA 4.6. *$\forall k \ge 1$, $d_t(E_k, C_k) = k$ and $d_t(0 \cdot E_k, 0 \cdot C_k) = k$.*

Proof. Both cases follow at once by application of Theorems 4.2 and 4.5. ◻

THEOREM 4.7. *$\forall n \ge 1$, $D_t(n) = \lfloor n/2 \rfloor$.*

Proof. This is an immediate consequence of Theorem 4.5 and Lemma 4.6. ◻

THEOREM 4.8. *Let S and T be related binary strings of length $2n \ge 4$. Then $d_t(S,T) = n$ if and only if S and T are isomorphic to C_n and E_n.*

Proof. We prove this theorem by induction. The base case is when $n = 2$. Then, by complete search, it may be verified that $d_t(S,T) = 2$ if and only if S and T are isomorphic to E_2 and C_2. Now suppose that the theorem holds when $n \le k$. Let S and T be strings of length $2k+2$ such that $d_t(S,T) = k+1$. We show that S and T are isomorphic to C_{k+1} and E_{k+1}.

By Lemma 4.4 we can apply a transposition to S or T that increases the combined length of the common prefix and suffix by at least two. Without loss of generality, we can relabel S and T so that the transposition found in the proof of Lemma 4.4 is applied to S resulting in the string S'. Furthermore, we can assume, without loss of generality, that $S(1) = 0$.

It must be that $lcp(S',T) + lcs(S',T) = 2$ and $d_t(S',T) = k$, since any alternative would contradict $d_t(S,T) = k+1$. Let S'_e and T_e be the strings S' and T excluding any common prefix and suffix. By the induction hypothesis, S'_e and T_e must be isomorphic to E_k and C_k. Therefore, either (a) $S'_e = E_k$ and $T_e = C_k$, or (b) $S'_e = C_k$ and $T_e = E_k$, or (c) $S'_e = {E_k}^R$ and $T_e = {C_k}^R$, or (d) $S'_e = {C_k}^R$ and $T_e = {E_k}^R$.

By the proof of Lemma 4.4, there are essentially two ways that the transposition can be applied to S, as typified by Cases (i) and (ii) in that proof. We take each case in turn and show that for the four possible values of S'_e and T_e, $d_t(S,T) = k+1$ if and only if S and T are isomorphic to E_{k+1} and C_{k+1}.

Case (i). $S = 0 \ldots 1$, $T = 1 \ldots 0$. In this case the transposition moves the block of zeros at the front of S to the end. We show that cases (c) and (d) for S'_e and T_e establish the induction step, whereas cases (a) and (b) lead to contradictions.

(a) $S' = 1 \cdot E_k \cdot 0 = 1 \cdot 0^k \cdot 1^k \cdot 0$ and $T = 1 \cdot C_k \cdot 0 = 1 \cdot (10)^k \cdot 0$. Therefore $S = 01 \cdot 0^k \cdot 1^k$. However, then the transposition $0[100][0^{k-2} \cdot 1^k]$ produces a string S'' such that $d_t(S'',T) < k$ by the induction hypothesis. Therefore $d_t(S,T) < k+1$, giving a contradiction.

(b) $S' = 1 \cdot C_k \cdot 0 = 1 \cdot (10)^k \cdot 0$ and $T = 1 \cdot E_k \cdot 0 = 1 \cdot 0^k \cdot 1^k \cdot 0$. Then $S = 001 \cdot (10)^{k-1} \cdot 1$. However, then the transposition $[001 \cdot (10)^{k-1}][1]$ produces a string

S'' such that $d_t(S'',T) < k$ by the induction hypothesis. Therefore $d_t(S,T) < k+1$, giving a contradiction.

(c) $S' = 1 \cdot {E_k}^R \cdot 0 = {E_{k+1}}^R$ and $T = 1 \cdot {C_k}^R \cdot 0 = C_{k+1}$. Therefore $S = E_{k+1}$.

(d) $S' = 1 \cdot {C_k}^R \cdot 0 = C_{k+1}$ and $T = 1 \cdot {E_k}^R \cdot 0 = {E_{k+1}}^R$. Therefore $S = {C_{k+1}}^R$.

Case (ii). $S = 0 \dots 11 \dots 0$, $T = 1 \dots 1$. In this case the transposition applied to S to obtain S' is of the form $[0^+(10^+)^*1][1 \dots 0]$. Note that this transposition splits the first occurrence of "11" in S. Note also that S' must contain "00" and end with "01," so only Case (c) needs to be considered. However, this cases leads to contradiction.

(c) $S' = 1 \cdot {E_k}^R \cdot 1 = 1 \cdot 1^k \cdot 0^k \cdot 1$ and $T = 1 \cdot {C_k}^R \cdot 1 = 1 \cdot (01)^k \cdot 1$. Then $S = 0^+ \cdot 1^{k+2} \cdot 0^+$. However, the transposition $0^+ \cdot 1^k[11][0^+]$ produces a string S'' such that $d_t(S'',T) < k$ by the induction hypothesis. Therefore $d_t(S,T) < k+1$, giving a contradiction.

Therefore $d_t(S,T) = k+1$ if and only if S and T are isomorphic to E_{k+1} and C_{k+1}. Therefore by induction the theorem is true. ∎

Again, we note that there appears to be no direct analogue of Lemma 4.4 when the alphabet size is > 2 and therefore no easy generalization of Theorem 4.7.

4.4. Sorting by transpositions.

We show that $d_t(S, S_\imath)$ can be determined in polynomial time. The following lemma can be verified easily.

LEMMA 4.9. *Let S' be a string obtained from S by a single transposition. Then*

$$z(S') \geq z(S) - 1.$$

With this lemma we can determine $d_t(S, S_\imath)$.

THEOREM 4.10. *For any binary string S,*

$$d_t(S, S_\imath) = \begin{cases} z(S) - 1 & \textit{if } S(1) = 0, \\ z(S) & \textit{otherwise.} \end{cases}$$

Proof. By Lemma 4.9, $d_t(S, S_\imath) \geq z(S) - 1$. If $S(1) = 0$, then $z(S) - 1$ transpositions of the form $0^+[1^+][0^+]1\dots$ or $0^+[1^+][0^+]$ transform S into $S_\imath$. If $S(1) = 1$, an extra transposition is required, because it is impossible to change the letter at the front of the string to 0 with a transposition and also reduce the value of z. The bound in this case can be achieved by performing the transposition $[1^+][0^+]1\dots$ or $[1^+][0^+]$, followed by the sequence of transpositions used when $S(1) = 0$. ∎

The distance described in Theorem 4.10 can be calculated easily in polynomial time. The question of whether, in general, the transposition distance between any two strings can be calculated in polynomial time remains open.

5. NP-completeness of reversal distance.

In this section, we prove that the general problem of finding the reversal distance between two related strings is NP-hard, even if the strings are drawn from a binary alphabet. We begin with a definition of the reversal distance problem as a decision problem (RD):

RD
Instance: Related strings S and T of length n, over an alphabet of size $t \geq 2$, and a bound $d \in Z^+$.
Question: Is $d_r(S,T) \leq d$?

The proof consists of a pseudopolynomial transformation from 3-Partition to RD. The definition of 3-Partition is as follows:

3-*Partition*
Instance: A set A of $3m$ elements, a bound $B \in Z^+$, and a size

$s(a) \in Z^+$ for each $a \in A$ such that $B/4 < s(a) < B/2$ and such that $\sum_{a \in A} s(a) = mB$.
Question: Can A be partitioned into m disjoint sets $A_1, A_2, \ldots, A_m$, such that, for $1 \leq i \leq m$, $\sum_{a \in A_i} s(a) = B$? (Note that each A_i must contain exactly three elements from A.)

Garey and Johnson [9] (see also Chapter 4.2 of [10]) have proved that 3-Partition is NP-complete (in the strong sense). Therefore a pseudopolynomial transformation from 3-Partition to RD is enough to prove that RD is NP-complete.

Theorem 5.1. *RD is NP-complete, even when* $t = 2$.

Proof. RD is in NP because, given a sequence of reversals, it can easily be checked in polynomial time that the sequence transforms S into T and has length at most d. We now describe the pseudopolynomial transformation from 3-Partition to RD.

Let I be an instance of 3-Partition. From this instance construct an instance, I', of RD with $S = (\sum_{i=1}^{3m-1} 0^{s(a_i)} 1^3) \cdot 0^{s(a_{3m})}$, $T = (0^B 1)^m \cdot 1^{8m-3}$, and $d = 3m - 1$. Therefore the blocks of zeros in S represent elements of A and the lengths of the blocks represent the sizes of the elements.

We first show that $d_r(S,T) \geq 3m - 1$.

Let U be a string that is related to S and T. Recall that $z(U)$ is the number of blocks of zeros in the string U. Define $o(U)$ as the number of blocks of ones of length one in U. Then the function $f(U) = z(U) - o(U) - 1$ can be viewed as a kind of distance function between strings U and T, since $f(T) = 0$. Furthermore, $f(S) = 3m - 1$. We show that, if U' is obtained from U by applying a single reversal, then $f(U') \geq f(U) - 1$.

Suppose that ρ is a reversal that transforms U into U' with $f(U') < f(U)$. Then ρ must reduce the number of blocks of zeros or increase the number of blocks of ones of length one.

If ρ reduces the number blocks of zeros, then it must have the form $\ldots 1[0 \ldots 1]0 \ldots$ or $\ldots 0[1 \ldots 0]1 \ldots$, and therefore $f(U) - f(U') = 1$. Therefore it is impossible for ρ to increase the number of blocks of ones of length one as well as reduce the number of blocks of zeros.

If ρ increases the number of blocks of ones of length one, then it must be of the form $\ldots 01[10 \ldots 0]0 \ldots$, or $\ldots 1[10 \ldots 11]0 \ldots$, or $\ldots 01[1 \ldots 0]11 \ldots$, or the mirror image of one of these three reversals. (Note the reversals $\ldots 01[11 \ldots 0]0 \ldots$ and $\ldots 11[10 \ldots 0]0 \ldots$ do not reduce the value of f.) In each case $f(U) - f(U') = 1$. Therefore $d_r(S,T) \geq 3m - 1$.

Note that the first reversal in the previous paragraph is special because it increases the number of blocks of length one by two but also increases the number of blocks of zeros. We call this kind of reversal a *bad reversal*.

We now show that the given transformation from 3-Partition to an instance of RD is a pseudopolynomial transformation. To do so we have to verify four standard properties of a pseudopolynomial transformation [10, p. 101]. We verify these four properties in turn.

For property (a), we have to show that I is a yes instance of 3-Partition if and only if $d_r(S,T) \leq 3m - 1$.

We have already shown that $d_r(S,T) \geq 3m - 1$. We now show that if $d_r(S,T) = 3m-1$, then no minimal length sequence of reversals that transforms S into T contains a bad reversal.

Suppose that $d_r(S,T) = 3m-1$. Every reversal in a minimal length sequence that transforms S into T must reduce the value of f by one. For a reversal to be bad the string must contain 0110 as a substring. However, S contains no such substring, and no reversal that reduces the value of f by one can create such a substring. Therefore if $d_r(S,T) = 3m-1$, then no minimal length sequence of reversals that transforms S into T contains a bad reversal.

This means that if $d_r(S,T) = 3m-1$, each block of zeros in T is constructed from three blocks of zeros in S. It follows that I is a yes instance of 3-Partition if $d_r(S,T) = 3m-1$.

Now we show that if I is a yes instance of 3-Partition, then $d_r(S,T) \leq 3m-1$. Since I is a yes instance, we can partition A into m disjoint sets $A_1, \ldots, A_m$, each of which contains three elements and sums to B. For each subset A_i in turn, we can use two reversals of the form $\ldots 0[1\ldots 0]a\ldots$, where $a \in \{1, \omega\}$ (where ω is the special character used to denote the end of the string) to merge the three blocks of zeros representing the elements of A_i into a single block of zeros of length B without affecting any other block of zeros. (Note that the reversals shown do not move the block of zeros at the front of the string.) Then we can use $m-1$ reversals of the form $\ldots 01[11\ldots 0]1\ldots$ to create blocks of ones of length one separating the blocks of zeros. This sequence of reversals has length $3m-1$, so $d_r(S,T) \leq 3m-1$. This establishes the required property (a).

To prove properties (b), (c), and (d) we need Length and Max functions for 3-Partition and RD. For 3-Partition, reasonable definitions are $\text{Length}(I) = |A| + \sum_{a\in A}\lceil \log_2 s(a)\rceil$ and $\text{Max}(I) = \max\{s(a) : a \in A\}$. For RD, reasonable definitions are $\text{Length}'(I') = 2n + \lceil \log_2 d\rceil$ and $\text{Max}'(I') = 1$ (since RD is not a number problem). Note that $n = \sum_{a\in A} s(a) + |A| - 3$. Given these functions, the required properties can be proved quite easily.

Therefore the transformation is a pseudopolynomial transformation and therefore RD is NP-complete. ∎

The transformation just described was obtained after several other simpler transformations had been shown to fail. An example is a potential transformation from sorting by reversals to *RD*. Given a permutation π, define strings $S = (\sum_{i=1}^{n-1} 0^{\pi(i)}1)\cdot 0^{\pi(n)}$ and $T = (\sum_{i=1}^{n-1} 0^{i}1)\cdot 0^{n}$. One might conjecture that determining the value of $d_r(S,T)$ must also determine the value of $d_r(\pi)$. However, if $\pi = 3142$, then $d_r(\pi) = 3$, but $S = 0001010000100$, $T = 0100100010000$, and $d_r(S,T) = 2$. Therefore this transformation does not work.

6. Conclusion.

In this paper we have shown that, just as sorting permutations by reversals is NP-hard, so also is finding the reversal distance between two strings, even when the strings are drawn from a binary alphabet. We have derived lower and upper bounds for the reversal distance between binary strings and used these to find the reversal diameter of $\mathcal{B}_n$.

The complexity of finding the transposition distance between two strings remains open, just as the complexity of sorting permutations by transpositions is open. We have also derived lower and upper bounds for the transposition distance between binary strings and used these to find the transposition diameter of $\mathcal{B}_n$. This contrasts with the problem of transposition diameter for permutations, which is unresolved.

In [6], Christie introduces the problem of sorting by block-interchanges. A block-interchange is similar to a transposition, except that the substrings that are swapped need not be adjacent. Christie proved that this problem could be solved in polynomial time. When extended to strings, however, it can be shown [8] in a manner similar to that used in the proof of Theorem 5.1 that the block-interchange distance problem is NP-hard, even when the strings are drawn from a binary alphabet.

REFERENCES

[1] V. Bafna and P. A. Pevzner, *Genome rearrangements and sorting by reversals*, SIAM J. Comput., 25 (1996), pp. 272–289.

[2] V. Bafna and P. A. Pevzner, *Sorting by transpositions*, SIAM J. Discrete Math., 11 (1998), pp. 224–240.

[3] P. Berman and S. Hannenhalli, *Fast sorting by reversals*, in Combinatorial Pattern Matching, Lecture Notes in Comput. Sci. 1075, Springer, Berlin, 1996, pp. 168–185.

[4] A. Caprara, *Sorting by reversals is difficult*, in Proceedings of the First International Conference on Computational Molecular Biology (RECOMB'97), ACM Press, Santa Fe, NM, 1997, pp. 75–83.

[5] A. Caprara, *Sorting permutations by reversals and Eulerian cycle decompositions*, SIAM J. Discrete Math., 12 (1999), pp. 91–110.

[6] D. A. Christie, *Sorting permutations by block-interchanges*, Inform. Process. Lett., 60 (1996), pp. 165–169.

[7] D. A. Christie, *A 3/2-approximation algorithm for sorting by reversals*, in Proceedings of the Ninth Annual ACM-SIAM Symposium on Discrete Algorithms, San Francisco, 1998, pp. 244–252.

[8] D. A. Christie, *Genome Rearrangement Problems*, Ph.D. thesis, Department of Computing Science, University of Glasgow, Glasgow, Scotland, 1998.

[9] M. R. Garey and D. S. Johnson, *Complexity results for multiprocessor scheduling under resource constraints*, SIAM J. Comput., 4 (1975), pp. 397–411.

[10] M. R. Garey and D. S. Johnson, *Computers and Intractability: A Guide to the Theory of NP-Completeness*, W. H. Freeman, San Francisco, 1979.

[11] S. Hannenhalli and P. A. Pevzner, *Transforming cabbage into turnip (polynomial algorithm for sorting signed permutations by reversals)*, in Proceedings of the 27th Annual ACM Symposium on Theory of Computing, Las Vegas, 1995, pp. 178–189.

[12] S. Hannenhalli and P. A. Pevzner, *Transforming cabbage into turnip: polynomial algorithm for sorting signed permutations by reversals*, J. ACM, 46 (1999), pp. 1–27.

[13] H. Kaplan, R. Shamir, and R. E. Tarjan, *Faster and simpler algorithm for sorting signed permutations by reversals*, in Proceedings of the Eighth Annual ACM-SIAM Symposium on Discrete Algorithms, New Orleans, 1997, pp. 344–351.

[14] J. Kececioglu and D. Sankoff, *Exact and approximation algorithms for sorting by reversals, with application to genome rearrangement*, Algorithmica, 13 (1995), pp. 180–210.