Bounding Prefix Transposition Distance for Strings and Permutations

Bhadrachalam Chitturi and I. Hal Sudborough

Department of Computer Science
Erik Jonsson School of Engineering and computer Science
University of Texas at Dallas
Richardson, Texas 75083

Abstract

A transposition is an operation that exchanges two adjacent substrings. When it is restricted so that one of the substrings is a prefix, it is called a prefix transposition. The prefix transposition distance between a pair of strings (permutations) is the shortest sequence of prefix transpositions required to transform a given string (permutation) into another given string (permutation). This problem is a variation of the transposition distance problem, related to genome rearrangements. An upper bound of n-1 and a lower bound of n/2 are known. We improve the bounds to n-$\log_8$ n and 2n/3 respectively. We also give upper and lower bounds for the prefix transposition distance on strings. For example, n/2 prefix transpositions are always sufficient for binary strings. We also prove that the exact prefix transposition distance problem on strings is NP complete.

1. Introduction

A permutation on $\Sigma = \{0,1,\ldots,n-1\}$ will be denoted by a list $x = (x_1, x_2, \ldots, x_n)$, where each element of Σ appears exactly once. A string on Σ will be denoted by a list $x = (x_1, x_2, \ldots, x_n)$, which may include repetitions of elements in Σ. Strings S and T on Σ are compatible if each symbol σ of Σ has the same frequency of occurrence in S and T. Several operations have been studied for transforming permutations and strings. For example, reversals [3,20], transpositions [2, 17], block moves [1], cut/paste moves [9], prefix reversals [14,16], short swaps (reversals of size at most three) [19], and prefix transpositions [10]. In this paper we give improved bounds for the prefix transposition distance on permutations and give new results for the prefix transposition distance on strings.

Bafna and Pevzner [2] studied transpositions and gave a lower bound of $\lfloor n/2 \rfloor + 1$ and an upper bound of *3n/4* for permutations of length *n*. They also gave a *3/2* approximation algorithm for sorting permutations by transpositions. Dias and Meidanis [10] studied the prefix transposition distance on permutations and showed the following: (a) an upper bound of *n*-1, (b) a lower bound of *n/2*, and (c) R_n, the reverse order permutation, can be sorted in 3*n*/4 steps. They conjecture that R_n is hardest.

The problem of transforming strings with finite alphabet size was studied by Christie and Irving [4] and later by Radcliffe et al [21]. Christie and Irving showed that both $d_r(S,T)$ and $d_t(S,T)$ are bounded above by *n/2*, where $d_r(S,T)$ (respectively, $d_t(S,T)$) is the minimum number of reversals (respectively, transpositions) necessary to transform a binary string *S* of length *n* into a compatible string *T*. They also showed that the problem of finding $d_r(S,T)$ for binary strings is NP-hard.

Radcliffe et al [21] showed that (a) the reversal distance, $d_r(S,T)$, between a pair of compatible *k-ary* strings of length *n* is at most $n - max\{a_i\}$ where $max\{a_i\}$ is the number of occurrences of the most frequent symbol, (b) the reversal distance between random strings, each having a positive fraction of every element, is $\theta(n/\log n)$, and (c) a ternary string can be optimally sorted by reversals. Radcliffe et al [21] also showed that the problem of finding the transposition distance and the signed reversal distance between two strings over a finite alphabet is NP-hard.

Hurkens et al [16] and Chitturi and Sudborough [6,7,8] gave bounds for prefix reversals on strings. For example, Hurkens et al gave the bound $d_{pr}(s,t) \leq 2(n-\alpha)$, where d_{pr} is the prefix reversal distance for strings of length n and α is the number of occurrences of the most frequent symbol. [16] and [8] show that the problem of finding $d_{pr}(S,T)$ for k-ary (k > 1) strings S and T is NP complete.

In section two we give improved lower bound and in section three we give an improved upper bound for the number of prefix transpositions needed to sort permutations. In section four we give an upper bound of

1530-1605 2008 U.S. Government Work Not Protected by U.S. Copyright

n-α for the number of prefix transpositions needed to transform a string S of length n into a compatible string T where α denotes the number of occurrences of the most frequent symbol. In section five we show that the problem of finding prefix transposition distance is NP complete.

A *transposition* on a list S relocates a sublist from its original location to a new location. For example; "it pours when it rains" becomes "when it rains it pours" in one transposition where the prefix "it pours" is relocated to the end. A transposition is defined by two parameters, namely the sublist that is being moved and its destination. Given a list $\pi = (\pi_1, \pi_2, \ldots, \pi_n)$ a transposition $\rho(i,j+1,k)$ repositions the sublist $(\pi_i, \pi_{i+1}, \ldots, \pi_j)$ at the destination between π_k and π_{k+1}. As a prefix transposition always moves a prefix, it need only describe the position just after the prefix and the destination. That is, $\rho(i+1, j)$ represents the prefix transposition on the list $\pi = (\pi_1, \pi_2, \ldots, \pi_n)$ that moves the prefix $(\pi_1, \pi_2, \ldots, \pi_i)$ to the destination between π_j and π_{j+1}. We write $\alpha \rightarrow \beta$ to denote that β is obtained from α by a single prefix transposition. Note that ρ(j-i+1,j) is the inverse of ρ(i+1,j). That is, successive moves ρ(i+1,j) and ρ(j-i+1,j) yield the original list. Given two lists π and μ, the prefix transposition distance between π and μ, denoted by $d_{pt}(\pi,\mu)$, is the length of a shortest sequence of prefix transpositions that transforms π into μ.

2. An Improved Lower Bound

Here we improve the lower bound for prefix transposition distance on permutations to *2n/3*. Let $\pi = (\pi_1, \pi_2, \ldots, \pi_n)$ be a permutation on {0,1, …, n-1}. Two adjacent integers π_i and π_{i+1} form an *adjacency* if $\pi_{i+1} = \pi_i + 1 \pmod n$ and form an *anti-adjacency* if $\pi_{i+1} = \pi_i - 1 \pmod n$. Let $(\pi_i, \pi_{i+1}, \ldots, \pi_j)$ denote the *sublist* of the permutation $(\pi_1, \pi_2, \ldots, \pi_n)$ from position i to position j. A *clan* is a maximal length sublist of anti-adjacencies. A *block* is a maximal length sublist of adjacencies. A *singleton* is a single element that does not form an adjacency with its neighbors. Singletons and blocks are called *objects*.

We denote the identity (i.e. sorted order) permutation (0,2,..,n-1) with I_n and the reverse order permutation (n-1, n-2, ..1,0) with R_n. I_n has one block of length n and no clans; R_n has one clan of length n and no blocks. In order to obtain I_n from R_n, one needs to eliminate all clans. We have defined adjacencies in a cyclic or wrap-around form, *i.e.* n – 1 is adjacent to 0. Therefore when a permutation has exactly n – 1 adjacencies it is either I_n or of the form (k,k+1,...n-1,0,..k-1), for some k. Notice that in one move one can transform the latter permutation into I_n. It should be noted that a block is deemed to be one object, so there is a decrease in the number of objects each time a new adjacency is made.

As a single prefix transposition can make up to two adjacencies, moves that make less than two adjacencies are viewed as incurring *waste*. If a move makes no adjacencies, then we say it makes two units of waste and a move that makes one adjacency makes one unit of waste. Even though a move can make two adjacencies, we will show that clans of length three or more are impediments and their removal requires steps that involve units of waste. As R_n is one clan of length n, any sorting of R_n must involve units of waste. Specifically, we use the concepts of *clan* and *waste* to derive a function Φ(π) which is a lower bound for the minimum number of moves required to sort any permutation π.

A permutation π allows a move that makes two new adjacencies, called a *double*, if and only if, for some j ($2 \le j \le n-1$), the first element $\pi_1 \equiv \pi_j + 1 \pmod n$ and, for some k ($1 \le k \le j-1$), $\pi_k \equiv \pi_{j+1} - 1 \pmod n$. In this case one can take the prefix consisting of everything from π_1 to π_k and reposition it between the elements π_j and π_{j+1}. (Note that this prefix is the first element alone if $\pi_1 \equiv \pi_{j+1} - 1 \pmod n$). For example, if π is the permutation *(3,5,4,2,6,1,0)* a double consists of moving the prefix consisting of the symbols *3,5* to the position between the symbols *2* and *6*, which creates the new permutation *(4,2,3,5,6,1,0)* with two new adjacencies. We observe that, if a permutation $\pi = \pi_1\, \pi_2 \ldots \pi_n$ allows a double, there is only one such move that is a double.

For an arbitrary permutation $\pi = \pi_1\, \pi_2 \ldots \pi_n$, a double is not always possible. For example, it is not possible in the permutation *3,0,2,6,5,1,4*. On the other hand, a move that makes one new adjacency, called a *single*, is always possible. For example, consider a permutation $(\pi_1, \pi_2, \ldots, \pi_n)$ on {0,1, …, n-1}. Let j ($2 \le j \le n$) be such that $\pi_1 \equiv \pi_j + 1 \pmod n$. Then, one can move any nonempty prefix of $(\pi_1, \pi_2, \ldots, \pi_{j-1})$ to the position following π_j and achieve at least one new adjacency. Also, let k and r ($1 \le k < r \le n$) be such that $\pi_k - 1 \equiv \pi_r \pmod n$. Then, one can move the prefix $(\pi_1, \pi_2, \ldots, \pi_k)$ to the position just before π_r and achieve at least one new adjacency. For example, for the permutation *(3,0,2,6,5,1,4)* there are many singles. One can move any nonempty prefix of *3,0* to the position just after *2*, move the prefix *3* to the position just before *4*, or move the prefix *3,0* to the position just before *1*.

Given a permutation π, let Γ(π) denote the collection of all clans of length at least three in π. Let

$$w(\pi) = \Big(\sum_{C \in \Gamma(\pi)} |C| - 2\Big)/3 .$$

The value of w(π) is intended as a measure of waste required in any sequence of moves that sorts the permutation π. In fact, we show that a lower bound on the number of steps to sort a permutation π is given by

the function $\Phi(\pi) = ½(|\pi| + w(\pi))$, where $|\pi|$ denotes the number of objects in π

Lemma 1. The total number of moves to sort a permutation π is at least $\Phi(\pi) \geq ½(|\pi| + w(\pi))$.

Proof. We show for any sequence of k steps, $k\geq 2$, $\Phi(\pi)$ decreases by no more than k. As $\Phi(I_n) = ½$, where I_n represents the identity permutation of length n, it follows from this that the number of steps to sort π is at least $\Phi(\pi)$.

First observe that, if one performs a prefix transposition that does not change $w(\pi)$, then $\Phi(\pi)$ decreases by at most one. That is, at most two adjacencies can be created in a single step and, hence, the number of objects in a permutation decreases by at most two, and so $\Phi(\pi)$ decreases by at most one. Observe also that a prefix transposition can alter clan structure, as it can:

(1) take a prefix that includes an initial portion of a clan, and

(2) insert a moved prefix inside a clan.

Breaking a clan C of length ≥ 3 into two clans C_1 and C_2, respectively, such that $|C|= |C_1|+|C_2|$, reduces $w(\pi)$ by at most ⅔. That is, the term $|C|-2$ is replaced by the sum $|C_1|-2 + |C_2|-2$, which is smaller by at most two. (Observe that the decrease can be less than two, as either C_1 or C_2 may be of length 1). As $w(\pi)$ is one third of the sum, $w(\pi)$ is smaller by at most ⅔. So, if a prefix transposition move does both (1) and (2) above, *i.e.* breaks one clan by including an initial portion in the chosen prefix and breaks another clan by inserting the prefix into it, then $w(\pi)$ can change by at most 2(⅔)=4/3. And, as $\Phi(\pi)$ decreases by one half of the decrease in $w(\pi)$, the change due to $w(\pi)$ in $\Phi(\pi)$ is at most ⅔.

Consequently, if a prefix transposition changes the clan structure only or makes new adjacencies only, then the decrease in $\Phi(\pi)$ is at most one. However, there is the possibility that a prefix transposition does both.

A prefix transposition may make new adjacencies and simultaneously change the clan structure. We shall say that a clan is broken at the *left end* (*right end*, respectively) by insertion, if a prefix is inserted into the clan just after the clan's initial symbol (just before the clan's final symbol, respectively). If a prefix is inserted into a clan, but not at an end, then no new adjacency can be created. That is, if a prefix $(y_1,\ldots, y_k)$ is inserted between successive clan elements x_{i+1} and x_{i+2}, to create the sublist $(x_1..x_i, x_{i+1}, y_1,\ldots, y_k , x_{i+2},.., x_p)$, $(i \geq 1, p \geq i + 3)$ then there is no adjacency between x_{i+1} and y_1, as $x_i = x_{i+1}+1$, and there is no adjacency between y_k and x_{i+2}, as $x_{i+3} = x_{i+2}-1$. So, to make an adjacency a clan must be broken at its left end or its right end. Moreover, if an insertion of a prefix P is made at the left end (respectively, right end) of a clan C, no adjacency can be made at P's right end (resp., left end).

The sum in $w(\pi)$ only includes clans of length at least three and one cannot break such a clan at both ends simultaneously. Hence, it follows that at most one adjacency can be made when a clan C of length at least three is broken by insertion of a prefix at an end. And, if a move makes an adjacency by inserting a prefix P at the end of a clan C, then the reduction in the length of C is one.

Also, if a clan C is broken by deletion of a prefix P, as described in (1) above, and an adjacency is made at the right end of P then the deletion must be at C's left end. That is, if a prefix α, $y_1,\ldots, y_i$ is moved and $(y_1,\ldots, y_i)$ is an initial portion of a clan, with $i\geq 2$, then $y_{i-1} = y_i+1$, so y_i can not make an adjacency with the new following element where it is moved.

Consider a move which breaks a clan C_1 by deletion, a clan C_2 by insertion, and makes one adjacency. When one adjacency is made, then the number of objects goes down by one, the insertion into C_2 makes its length go down by 1, and the deletion of part of C_1 makes the sum of clan lengths go down by at most two. Consequently, the maximum decrease in $\Phi(\pi)$ with such a move is $½(1 + 1/3(1+2)) = 1$. So, the reduction of $\Phi(\pi)$ is bounded above by the number of moves. Other prefix transposition moves that make one adjacency will make smaller changes in clan structure and, hence, $\Phi(\pi)$ will go down by less than 1. So, for all moves that make one new adjacency, the reduction of $\Phi(\pi)$ is bounded above by the number of moves.

There is one possible move for a given permutation π that can simultaneously make two new adjacencies and reduce $w(\pi)$. It is a move that takes a prefix, let us say starting with p and ending with e, where e is an initial element of a clan C of length at least three, and moves it to a position between p-1 (mod n) and e+1 (mod n). This move makes two adjacencies and also reduces the length of a clan by one. In other words, it reduces $\Phi(\pi)$ by $½(2 + 1/3(1)) = 7/6$, call this *m*. However, as we shall see, any succeeding move, say *m'*, reduces $\Phi(\pi)$ by at most 5/6, so *m* and *m'* together reduce $\Phi(\pi)$ by at most 7/6+5/6 = 2. Thus, even here, the decrease in $\Phi(\pi)$, namely 2, is bounded by the number of moves.

Look again at the move *m* described in the previous paragraph. It moves a prefix that ends with the initial element e of a clan C of length at least three. Consequently, the remainder of the clan C becomes an initial sublist of the resulting, new permutation. So this new permutation has a clan of size at least two as a prefix. In such a case, consider a move *m'* on this new permutation. For *m'* there is one possibility of making a new adjacency at the left end of a moved prefix, namely the prefix is just the first element of the clan and it is inserted immediately after the second element of the

clan. This reduces the size of the clan by two and makes one adjacency. So, the change in $\Phi(\pi)$ is ½ + ½(2/3) = 5/6. So, the claim that the next move reduces $\Phi(\pi)$ by at most 5/6 is true in this case.

Any other choice for *m'* can only make an adjacency at the right end of the moved prefix. If no adjacency is made in the next move, then, as we have seen, the decrease in $\Phi(\pi)$ is at most ⅔ < 5/6. If an adjacency is made, then it must be that the chosen prefix through its deletion either (a) does not break a clan of length at least 3 or (b) it breaks a clan of length at least 3 by including its first element. In both cases (a) and (b), the decrease in $w(\pi)$ is at most 1/3, as the only other possible change in clans, as the move is assumed to make a new adjacency, is that a clan is broken by insertion of this prefix into its right end. So, $\Phi(\pi)$ decreases by at most ½(1 + 1/3(1+1)) = 5/6. ■

Theorem 1. The lower bound to sort R_n is 2n/3 moves.

Proof. $\Phi(I_n)$ = ½ and by Lemma 1, the number of moves required to sort a permutation π is at least $\Phi(\pi) \geq ½(|\pi| + w(\pi))$. For the permutation $\pi = R_n$ we have $|\pi| = n$ and $w = (n-2)/3$. This yields $\Phi(R_n) = n/2 + 1/6(n–2) = n/2 + n/6 -1/3 = 2n/3 -1/3$. (Note that the last move can not be the one that reduces $\Phi(\pi)$ by 7/6 (m), since *m* always makes a clan appear in front and the identity permutation has no clans.) Therefore the lower bound is $\Phi(R_n) - \Phi(I_n) = 2n/3 -1/3 - 1/2 = 2n/3$.■

3. An Improved Upper Bound

A greedy strategy of always choosing a move that makes a maximum number of adjacencies is not always optimum. For example, Dias et al [10] have shown that, for all n≡0 (mod 4), the permutation R_n, can be sorted with a sequence of 3n/4 prefix transpositions. Their sorting algorithm for R_n begins with moves that make no adjacencies, although as we have seen single adjacency moves are always possible. For example, the Dias et al sorting sequence for R_8 is as follows:

(7,6,5,4,3,2,1,0)
➔ (3,2,1,7,6,5,4,0) (no new adjacency)
➔ (1,7,6,3,2,5,4,0) (no new adjacency)
➔ (6,3,2,5,4,0,1,7)
➔ (2,5,6,3,4,0,1,7)
➔ (3,4,0,1,2,5,6,7)
➔ (0,1,2,3,4,5,6,7)

We observe that this sorting sequence makes seven adjacencies in six moves. Furthermore, a similar sequence of prefix transpositions makes seven adjacencies in six moves for any permutation of the form R'_8 = (7,α,6,5,4,3,2,1,0,β), where α and β are arbitrary sublists. That is, we can use the Dias et al sorting sequence given above, but with α added just after the element 7 and β added at the end. More generally, for any i, one can use the Dias et al sequence on a permutation of the form R'_8 = (7+i,α,6+i,5+i,4+i,3+i,2+i,1+i,i,β), and make 7 adjacencies in 6 moves.

We now extend this and show that, for any permutation $\pi = (\pi_1,\pi_2, \ldots,\pi_n)$ there is a sequence of k moves that makes at least k+1 new adjacencies, where k is less than or equal to 7(n-3)/8. Such a sequence is obtained by a greedy strategy; where at each step we make at least one new adjacency, unless the permutation is of the form R'_8, where one can instead use the Dias et al sequence to achieve seven adjacencies in six steps.

If we assume for the moment that our sequence of steps does not produce a permutation of the form R'_8, then we show there is a sequence of at most k-1 moves making single adjacencies followed by a move that makes two adjacencies. If we do produce a permutation of the form R'_8, then we make seven adjacencies in six moves by the Dias et al sequence, and hence we achieve k+1 adjacencies in k moves.

Recall in Section 2 we defined cyclic or "wrap-around" adjacencies. That is, n-1 is adjacent to 0.

Let $\pi = (\pi_1,\pi_2, \ldots,\pi_n)$ be an arbitrary permutation, where, for convenience, we let **x** and **y** denote the objects at the end (π_{n-1} and π_n respectively). If the first element π_1 in the permutation were x+1 (mod n), then one could immediately make two adjacencies by moving the prefix up to, and including, the element y-1 (mod n) into the position between x and y. (In the stated case, y-1 (mod n) must be somewhere between the first element x+1 (mod n) and x, since the elements x and y are at the end). We note that x is not y-1 (mod n); otherwise, x and y would be part of a block and not separate objects.

For any elements p and q in $\sum = \{0,1,\ldots,n-1\}$, let distance(p,q) = p-q (mod n). If the first element π_1 is not x+1 (mod n), then our strategy is to make an adjacency, by moving some prefix to a position immediately after the element π_1-1 (making a new adjacency between π_1-1 and π_1), and to choose this prefix in such a way that it makes the new first element as close as possible to x+1 (mod n). If one chooses the prefix containing everything up to, but not including the element π_1-1, then the new first element is π_1-1, and distance(π_1-1,x+1 mod n) is one less than the previous distance(π_1,x+1 mod n). In general, one chooses a prefix $(\pi_1,\pi_2, \ldots,\pi_{k-1})$ such that distance(π_k,x+1 (mod n)) = min { distance(π_j,x+1 (mod n)) | $(\pi_1,\pi_2, \ldots,\pi_j)$ is a proper prefix of $(\pi_1,\pi_2, \ldots,\pi_1-1)$}. As the prefix ending with the element just before π_1-1 is one choice, the distance to x+1 (mod n) decreases by at least one.

Algorithm:

In a given permutation obtained at any step, if a double is possible then execute it and stop. If there is a prefix of the form R'_8 in the permutation obtained, then sort it using the Dias et al sequence and stop. Otherwise, denote the first element of the permutation by *f* and do the step indicated in the cases described below:

Case (1). f = x+1 (mod n)

Take the prefix (f,…, y-1 (mod n)) and move it to the position between x and y making two new adjacencies.

Case(2). f = y+1 (mod n)

(A) x+1 (mod n) is before y-1 (mod n).
Let p be the element immediately preceding x+1 (mod n); take the prefix (f,..,p) and move it to the end (after y) making one new adjacency.

(B) y-1 (mod n) is before x+1 (mod n).
Let p be the element just before y-1 (mod n). Let t be such that distance(t,x+1 (mod n)) is the minimum of {distance(π_i, x+1 (mod n)) | (π_1, π_2, … , π_i) = (f, … , p) }. Take the prefix (f,..,t') and move it to the end (after y) making one new adjacency, where t' is the element immedialtely preceding "t".

Case(3). f ≠ x+1 (mod n) and f≠ y+1 (mod n).

(A) f-1 (mod n) occurs after x+1 (mod n).
Let p be the element just before x+1 (mod n). Move the prefix (f,..p) to just after f-1 (mod n) making one new adjacency.

(B) f-1 (mod n) occurs before x+1 (mod n)
Let p be the element just before f-1 (mod n). Let t be such that distance(t,x+1 (mod n)) is min{distance(π_i, x+1 (mod n)) | (π_1, π_2, … , π_i) = (f, … , p) }. Move the prefix (f,..,t') to just after f-1 (mod n), making one new adjacency, where t' is the element immedialtely preceding "t".

For any permutation $\pi = \pi_1\ \pi_2 \ldots \pi_n$, there is a unique next permutation, denoted by N(π), obtained by the procedure described above, until either (a) x+1 (mod n) becomes the first element, or (b) a double can be made, or (c) a prefix of the form R'_8 occurs. Assume that neither (b) nor (c) occur. That is, the procedure produces a sequence of permutations π, N(π), N(N(π)) = $N^2(\pi)$, $N^3(\pi)$, … , $N^t(\pi)$, terminating with the permutation $N^t(\pi)$ where x+1 becomes the first element. Let the sequence of first elements of these permutations be denoted by $f^{(0)}$, $f^{(1)}$, $f^{(2)}$, … , $f^{(t)}$. By looking at the steps defined, it can be verified that, for all i (1≤i<t), $\text{distance}_n(f^{(i)},x) > \text{distance}_n(f^{(i+1)},x)$. That is, in the permutations produced, the first element gets closer and closer to the desired element, namely x+1 (mod n). Elements in the permutation π that do not appear in the sequence $f^{(0)}$, $f^{(1)}$, $f^{(2)}$, … , $f^{(t)}$ are called *unvisited elements*.

Furthermore, the sequence $f^{(0)}$, $f^{(1)}$, $f^{(2)}$, … , $f^{(t)}$ is a subsequence of (π_1, π_2 , … , π_n). That is, each step involves moving a prefix and no element of a prefix moved ever become an initial element again. (This follows from the fact that prefixes are chosen to include only elements that have greater distance to x than the new first element and the fact that after each step the distance between successive first elements and x decreases.) Specifically, in cases (2b) and (3b) we chose a prefix to ensure that the first element is the one that is closest to x+1 (mod n).

It follows that the original permutation $\pi = \pi_1,\pi_2,..,\pi_n$ can be written as $f^{(0)},\alpha_1,f^{(1)},\alpha_2,f^{(2)},\ldots,\alpha_t,f^{(t)},\alpha_{t+1}$, where α_{t+1} ends with the elements x, y, and, for all i (1≤i≤t+1), α_i is a sequence of unvisited elements. Furthermore, for any i (1≤i≤t), it can be seen that $N^i(\pi)$ can be written as $f^{(i)}$, β_{i+1}, $f^{(i+1)}$, α_{i+2}, $f^{(i+2)}$,…, α_t, $f^{(t)}$, β_{t+1}, where β_{i+1} and β_{t+1}, respectively, consist of all of the elements in α_i and α_{t+1}, respectively, together with some of the elements in $\gamma = f^{(0)}$, α_1, $f^{(1)}$, α_2, $f^{(2)}$, … , α_{i-1}, $f^{(i-1)}$, α_i , so that all of the elements in γ are either in β_{i+1} or in β_{t+1}.

To see this, we proceed by induction on i. The basis step, when i=0, is clearly true. For the inductive step, assume that $N^i(\pi) = f^{(i)}$, β_{i+1}, $f^{(i+1)}$, α_{i+2}, $f^{(i+2)}$, … , α_t, $f^{(t)}$, β_{t+1}. If $f^{(i)}$ = y+1 (mod n), so Case (2) applies, then $N^{i+1}(\pi) = f^{(i+1)}$, α_{i+2}, $f^{(i+2)}$, α_{i+3}, $f^{(i+3)}$, … , α_t, $f^{(t)}$, β'_{t+1}, where all of $f^{(i)}$, β_{i+1} has been added to β_{t+1} (immediately after the element y) to make β'_{t+1}. If $f^{(i)}$ = r, where r ≠ y+1 (mod n) and r ≠ x+1 (mod n), so Case (3) applies, then $N^{i+1}(\pi) = f^{(i+1)}$, β_{i+2}, $f^{(i+2)}$, α_{i+3}, $f^{(i+3)}$, … , α_t, $f^{(t)}$, β_{t+1}, where all of $f^{(i)}$, β_{i+1} has been added to α_{i+2} (immediately after the element r+1 (mod n)) to make β_{i+2}. So, the property follows by induction.

As observed earlier, for any permutation of the form R'_8 = 7+j, α, 6+j, 5+j, 4+j, 3+j, 2+j, 1+j, j β there is a sequence of 6 moves that makes 7 new adjacencies. Consequently, for any i and j, if $N^i(\pi)$ = 7+j, α, 6+j, 5+j, 4+j, 3+j, 2+j, 1+j, j, β, for some α and β, one can make 7 new adjacencies in 6 moves. That is, one can do better than the greedy strategy of achieving one adjacency per step. On the other hand, if $N^i(\pi)$, for all i, does not have this form, then it follows that either
(a) the sequence $f^{(i)}$, $f^{(i+1)}$, $f^{(i+2)}$, $f^{(i+3)}$, $f^{(i+4)}$, $f^{(i+5)}$, $f^{(i+6)}$, $f^{(i+7)}$ is not a consecutive sequence of decreasing integers, so some element is skipped, or
(b) there is at least one new unvisited element between the elements of the sequence $f^{(i+1)}$, $f^{(i+2)}$, $f^{(i+3)}$, $f^{(i+4)}$, $f^{(i+5)}$, $f^{(i+6)}$, $f^{(i+7)}$. It follows that either (a) one can make 7

adjacencies in 6 steps somewhere or (b) the length of the sequence of visited elements until x+1 (mod n) becomes the first element, is at most 7n/8. That is, in (b), for every sequence of eight visited elements, there is at least one unvisited element.

The successor chain from f to x+1 (mod n) has length of at most *n-3*. In cases 1, and in sub cases 2a and 3a, one either has x+1 (mod n) at the front or one makes a move so that this happens. As indicated earlier, when x+1 (mod n) is at the front there is move that makes two adjacencies, *i.e.* there is a double. By the above, we have shown that x+1 (mod n) comes to the front in at most *7/8(n-3)* moves, so we have the following:

Theorem 2 An upper bound for sorting permutations of length *n* is $n\text{-}\log_8 n$.

Proof. In the worst case we get the recurrence relation for sorting a permutation of size *n* as $T(n) = 7/8\ (n-3) + T(n-(7/8\ (n-3))-1) = 7/8\ (n-3) + T(n/8 + 13/8)$, where $7/8\ (n-3)$ are the number of moves executed. The size of remaining permutation is at most $(n-(7/8\ (n-3))-1)$, where the one that is subtracted corresponds to the double which we are guaranteed to make. Simplifying $T(n) = 7n/8 + T(n/8 + 13/8) - 21/8$. Since the problem size can be reduced by *13/8* by consuming at most *13/8* moves, the recurrence can be rewritten as: $T(n) = T(n/8) + 7/8n - 1$. This recurrence yields the solution of $T(n) = n - \log_8 (n)$.

4. Bounds for strings.

We consider compatible strings S and T. Let $S=s_1s_2...s_{n-1}s_n$ and $T=t_1t_2\ldots t_{n-1}t_n$ and "a" be the most frequent symbol.

Theorem 3. Let α be the number of times the most frequent symbol occurs in compatible strings S and T. Then $d_{pt}(S,T) \leq n-\alpha$.

Proof. The bound holds trivially for strings S and T of length 2. That is, if α =2, then S=T and no move is needed, and if α =1, then at most one move is needed. So, let $S=s_1s_2...s_{n-1}s_n$ and $T=t_1t_2\ldots t_{n-1}t_n$ be compatible strings over Σ of length n>2, and assume for an inductive hypothesis that the bound holds for all pairs of compatible strings of length less than n. Let a be the most frequent symbol in S (and, hence, also in T). If $s_n = t_n = a$, then $s_1s_2...s_{n-1}$ and $t_1t_2\ldots t_{n-1}$ are compatible strings of length n-1 where a occurs α – 1 times and the problem size reduces by 1. Thus, by the inductive hypothesis, $d_{pt}(S,T) \leq ((n-1)-(\alpha-1)) = (n-\alpha)$. If $s_n = t_n \neq a$, the inductive hypothesis gives $d_{pt}(s,t) \leq (n-1) - \alpha = n - \alpha - 1$. Suppose $s_n \neq t_n$, and assume $s_n = a$ and $t_n = b$, where a and b are different symbols. Then in one prefix transposition one can move an occurrence of the symbol b to the end of s, so s_n matches t_n and the prefixes of length n - 1 still have α "a"s. Hence by induction $d_{pr}(s,t) \leq 1 + (n-1-\alpha) = n - \alpha$.

Corollary 1. Theorem 3 yields $d_{pt}(S,T) \leq n/2$ for binary strings, $d_{pt}(S,T) \leq 2n/3$ for ternary strings, $d_{pt}(S,T) \leq 3n/4$ for quaternary strings.

We define the prefix transposition breakpoint distance between two strings S and T *i.e.* $b_{pt}(S,T)$, similar to the definition given in [4]. We extend the lengths of S and T by two; first we put in the front of S and T a special symbol □ and then we append a special symbol ■ at the end of S and T. These special symbols are not involved in any moves and are employed to measure the similarity between strings. Prefix transpositions do not change the order of elements therefore a '01' adjacency is different from a '10' adjacency. The function $f_{ab}(S)$ represents the number of '*ab*' adjacencies in the string S, where a and b are drawn from the set ∑', where ∑' = ∑ ⋃ {□, ■}. Each of the strings has a total of *n - 1* adjacencies. We are interested in measuring the adjacencies that are in S and not in T. Therefore we use the function δ(x), where δ(x)= x if x > 0, else δ(x) = 0.

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

Notice that this definition yields $b_{pt}(S,T) = b_{pt}(T,S)$. If the common adjacencies between S and T are *q* in number then S has *n-1-q* adjacencies that are not in T and vice versa. If S=T then $b_{pt}(S,T)=0$, but the converse is not true. For example, consider the strings S = 00011001 and T = 0011001. In this case we have $f_{□0}(S) = f_{□0}(T) = 1$, $f_{00}(S) = f_{00}(T) = 3$, $f_{01}(S) = f_{01}(T) = 2$, $f_{10}(S) = f_{10}(T) = 1$, $f_{11}(S) = f_{11}(T) = 1$, and $f_{1■}(S) = f_{1■}(T) = 1$ for the non zero values of f_{ab} and hence $b_{pt}(S,T)= 0$.

Lemma 2. If S' is obtained from S by a single prefix transposition then $b_{pt}(S',T) + 2 \geq b_{pt}(S,T) \geq b_{pt}(S',T) - 2$.

Proof: A move involves a cut (moved prefix) and a paste (insertion of the moved prefix). A cut can break an adjacency, but cannot make any adjacencies. A paste can make at most two adjacencies on either end of the substring being relocated or it can break an adjacency that existed before the paste. Therefore, $b_{pt}(S',T) + 2 \geq b_{pt}(S,T) \geq b_{pt}(S',T) - 2$. .■

Theorem 4. Let S and T be compatible strings of length n. Then $d_{pt}(S,T) \geq b_{pt}(S,T)/2$.

Proof. Lemma 2 shows that each move can reduce the number of breakpoints by at most two. This theorem directly follows from Lemma 2.■

Consider the strings S=(012012) and T=(002211). They do not have any common adjacencies therefore $b_{pt}(S,T) = 5$, which yields $d_{pt}(S,T) = 3$. In fact, three moves are required: [01]20*12 →[20011]2*→[22]00*11 →002211.

5. Transforming strings is NP-complete.

Garey and Johnson [13] proved that the 3-Partition problem is strongly NP-complete. Given a source string S, a target string T, and a sequence of prefix transpositions verifying that the given sequence transforms S to T can be done in polynomial time. Therefore, the prefix transposition problem is in NP.

In this section we prove that finding the prefix transposition distance between two compatible strings is NP-complete. The prefix being moved is indicated by square brackets and the destination position by an asterisk. We define the binary prefix transposition distance as a decision problem (BPT).

3Partition

Instance: A set of 3m elements, a bound B and a positive size $s(b_i)= a_i$, for each b_i in A, such that $B/4 < s(b_i) < B/2$ and $\sum_{b_i \in A} a_i = mB$.

Question: Can A be partitioned into m disjoint sets $A_1, A_2, .. A_m$ such that, for all i, $\sum_{j \in I_i} a_j = B$?, (where I_i is the set of indices of A_i). That is, the sum of all the elements of a set A_i is equal to B.

Binary prefix transposition distance problem (BPT)
Instance: Compatible binary strings S and T and an integer d.
Question: Is $d_{pt(S,T)} \leq d$?

Theorem 5. BPT is NP-complete.

***Proof*:** Let I be an instance of 3Partition; where the integer bound is B, $A = \{b_1, b_2, \ldots b_{3m}\}$, and for all the indices "i" the value $s(b_i) = a_i$. From I we construct an instance I* of BPT. Notice that $a_i = s(b_i)$ is the value of an element in the 3Partition problem and in I* it will be converted into an input parameter whose length is a_i. Therefore the reduction can be done in polynomial time only for small integer values. As 3Partition is known to be strongly NP-complete, we know that it remains NP-complete for small integers.

Let I* be the instance of BPT where

$$S = 10^{a_1} 10^{a_2} .. 10^{a_{3m-1}} 10^{a_{3m}} (10)^m$$

$$T = (1^4 0^{B+1})^m$$

$d = 3m$.

Where each 0 block in S represents an element b_i of 3Partition and the number of 0s equals a_i. Let "$10^{a_1} 10^{a_2} .. 10^{a_{3m-1}} 10^{a_{3m}}$" be the head of S and $(10)^m$ be the tail of S. All the m pairs of original 10s in the tail of S are receptacles. An unused receptacle is the one that is not involved in a prefix transposition. A singly used receptacle is the one that is involved in one prefix transposition and doubly used receptacle is the one that is involved in two prefix transpositions.

We will show that 3m moves are necessary. S has no 11 adjacencies, let us compute the number of 00 adjacencies in S. We compute this by first computing the number of 00 adjacencies in 0^{mB} and then, as the block is split into 3m pieces subtracting 3m -1. The number of 00 adjacencies in 0^{mB} is mB-1, so the total number of 00 adjacencies in all 0 blocks of S put together is mB-1 – (3m-1) = mB -3m.

Consider the adjacencies in T. The number of 00 adjacencies in 0^{B+1} is B, therefore the number of 00 adjacencies in $(0^{B+1})^m$ is mB. The total number of 11 adjacencies in 1^4 is 3, therefore the total number of 11 adjacencies in $(1^4 0^{B+1})^m$ is 3m. T has 3m more 00 adjacencies and 3m more 11 adjacencies than S. Therefore T has 6m different adjacencies than S. Each prefix transposition can increase the number of adjacencies by at most two; therefore 3m moves are necessary.

We complete the proof of the theorem by showing a reduction from 3Partition. This is done by showing that S can be transformed into T if and only if the instance I of 3Partition has a solution.

(a) If S can be transformed into T by 3m prefix transpositions then 3P has a solution: Since we accomplish this in the minimum possible moves, no 0-block can be split. Notice that for all values of i, $a_i > B/4$ and a_i is an integer. Therefore any four blocks of zeros put together will form a block which has at least B+4 zeros. Also, if two 0 blocks of S and one "10" are merged together with prefix transpositions, then the total number of zeros they can produce it at most B - 1. Therefore, each 0 block in T is formed from exactly three 0 blocks of S. So, we can list the moves made by BPT and observe which 0 blocks corresponding to which three a_i's are combined together, m such triples give the solution for 3P.

(b) If 3P has solution then S transforms to T in 3m moves: We give an algorithm that transform S into T in 3m moves.

An algorithm to transform S into T in 3m moves:

Let $a_{j1} + a_{j2} + a_{j3} = B$, we define the components $10^{a_{j1}}, 10^{a_{j2}}$, and $10^{a_{j3}}$ to be partners of each other. The components of the following pairs are also partners of each other: $\{10^{a_{j1}}, 10^{a_{j2}+a_{j3}}\}$, $\{10^{a_{j2}}, 10^{a_{j1}+a_{j3}}\}$ and $\{10^{a_{j3}}, 10^{a_{j1}+a_{j2}}\}$. We present the steps of the algorithm below.

1) Execute the following move.
$[10^{a_1}]10^{a_2}1\ldots10^{a_{3m}}\ (1\,1^+0^+)^+1*0\,(10)^{m-1} \rightarrow$
$10^{a_2}1\ldots10^{a_{3m}}\ (1\,1^+0^+)^+1^2*0^{a_1+1}\,(10)^{m-1}$, where the leading component is moved into an unused receptacle.

2) Execute the following step for 3m-iterations:
Let the leading component (leftmost) of the string be 10^{a_i}, where $B/4 < a_i < B/2$ (by definition of 3P). There are two cases:

a) None of the partners of 10^{a_i} are transposed. In other words, they were not prefixes of some earlier move. In this case, transpose 10^{a_i} into the leftmost unused receptacle. This move is shown below.
$[10^{a_i}]10^{a_{i+1}}1\ldots10^{a_{3m}}\ (1\,1^+0^+)^+1*0\,(10)^k \rightarrow$
$10^{a_{i+1}}1\ldots10^{a_{3m}}\ (1\,1^+0^+)^+1^2 0^{a_i+1}\,(10)^k$

b) At least one partner of the leading component is moved.

I. One partner of 10^{a_i} say 10^{a_j} is transposed into a receptacle. Execute the prefix transposition shown below.
$[10^{a_i}]10^{a_{i+1}}..10^{a_{3m}}(1^+0^+)^*1^2*0^{a_j+1}(1^+0^+)^*(10)^k$
$\rightarrow 10^{a_{i+1}}..10^{a_{3m}}(1^+0^+)^*1^3 0^{a_i+a_j+1}(1^+0^+)^*(10)^k$,
where the leading component is moved into a singly used receptacle.

II. Both partners are moved. Say 10^x is the partner of 10^{a_i}, where $x = B - a_i$. Execute the prefix transposition shown below.
$[10^{a_i}]10^{a_{i+1}}..10^{a_{3m}}(1^+0^+)^*1^3*0^{x+1}(1^+0^+)^*(10)^k \rightarrow$
$10^{a_{i+1}}..10^{a_{3m}}(1^+0^+)^*1^4 0^{B+1}(1^+0^+)^*(10)^k$, where the leading component is moved into a doubly used receptacle.

Parts (a) and (b) together prove that BPT is NP complete and hence the theorem is proved.

6. Approximation scheme

In this section we examine the problem of finding an approximation algorithm to transform a given string into another compatible string A *partition* of a string S is a sequence $S^p = (S_1, S_2, \ldots S_k)$ of substrings of A such that concatenation of $(S_1, S_2, \ldots, S_k)$ equals S. Given a partition S^p of a string S and a partition T^p of a string T, if S^p and T^p are permutations of each other then the pair (S^p, T^p) constitutes a common partition for S and T. The common partition of minimum cardinality yields a minimum common string partition (MCSP). The signed minimum common string partition (SMCSP) is a variation of MCSP in which each element of a string has an associated positive or negative sign. When a signed segment is reversed all the elements in the segment change their signs. Goldstein et al [12] showed that signed and unsigned versions of Minimum Common String Partition Problem (MCSP) are NP-hard if the symbols repeat two or more times

Claim 1: Let *O* be the optimum solution and '*p*' be the number of moves it takes. If the cardinality of MCSP of S and T is "*q*" then $(q-1)/2 \le p \le q-1$

Proof. In *p* moves a maximum of *2p* adjacencies and a minimum of $p - 1$ adjacencies can be made. We need to make $q - 1$ new adjacencies to transform S into T. If the optimum solution makes only doubles then we have $p = (q-1)/2$ and if it makes only singles then we have $p = q - 1$. Notice that singles are always possible. Therefore, $(q-1)/2 \le p \le q-1$.■

Cormode and Muthukrishnan [5] gave an $O(\log n \log^* n)$ approximation algorithm for String Edit Distance Matching Problem with Moves. In this problem the moves that are allowed are: Insert a character, Delete a character and Move a substring. Shapira and Storer [22] showed that eliminating the insert and delete character operations does not change the edit distance of two strings by more than a constant multiplicative factor. Therefore Cormode and Muthukrishnan [5] provides an $O(\log n \log^* n)$ approximation algorithm for MCSP. This result in conjunction with Claim 1 yields an *$O(\log n \log^* n)$ approximation algorithm* for the transformation of strings by prefix transpositions.

7. Conclusions

We gave bounds of $n\text{-}\log_8 n$ and $2n/3$ for sorting permutations. We also presented an upper bound of n-α steps for transforming a string S into a compatible string T, where α is the number of occurrences of the most frequent symbol in S and T. Specifically, this yields an upper bound of n/2 for binary strings, 2n/3 for ternary strings, etc. We also proved that the exact prefix transposition distance problem for strings is NP complete. We believe that *3n/4* moves are required to sort R_n.

8. References

[1] W.W. Bein, L. L. Larmore., S. Latifi, I. H. Sudborough. Block sorting is hard. Parallel Architectures, Algorithms and Networks, I-SPAN '02 Proceedings. International Symposium, p. 307-312, 2002.

[2] V. Bafna and P. Pevzner, Sorting Permutations by Transpositions, SIAM J. Discrete Mathematics, V.11, No. 2, p. 224-240,1998

[3] V. Bafna and P. A. Pevzner. Genome Rearrangements and Sorting by Reversals. Annual Symposium on Foundations of Computer Science, p.148-157,1993.

[4] D. Christie and R. Irving, Sorting Strings by reversals and transpositions. SIAM J. Discrete Math, v.14, no.2, p.193-206, 2001.

[5] G. Cormode and S Muthukrishnan. The String Edit Distance Matching Problem with Moves. Symposium on Discrete Algorithms; Proceedings of Thirteenth Annual ACM-SIAM Symposium on Discrete Algorithms; p.667-676, 2002.

[6] B. Chitturi and I. H. Sudborough. Prefix Transpositions on Strings. Dept. of Comp. Sci., University of Texas at Dallas, 11/06.

[7] B. Chitturi and I. H. Sudborough. Prefix Reversals on Strings. Dept. of Comp. Sci, University of Texas at Dallas, 11/06.

[8] B. Chitturi and I. H. Sudborough. On Complexity of Transforming Strings with Prefix Reversals and Prefix Transpositions. Dept. of Comp. Sci, University of Texas at Dallas, 11/06.

[9] D. Cranston, I. H. Sudborough, and D. B. West. Short Proofs for Cut-and-Paste Sorting of Permutations. “arXiv:math/0605084v1” [math.CO].

[10] Zanoni Dias and Joao Meidanis, Sorting by Prefix Transpositions, Lecture Notes in Computer Science, Springer Berlin/Heidelberg, v.2476/2002, p. 463-468, 2002.

[11] H. Eriksson, K. Eriksson, J. Karlander, L. Svensson, J. Wastlund, "Sorting a bridge hand", Discrete Mathematics, v.241, 2001, p.289-300.

[12] A. Goldstein, P. Kolman and J. Zheng. Minimum Common String Partition Problem: Hardness and Approximations. Lecture Notes in Computer Science, Springer Berlin / Heidelberg, ISSN: 0302-9743, Volume 3341, p.484-495, 2004.

[13] M. R. Garey and D. S. Johnson, “Strong” NP-Completeness Results: Motivation, Examples, and Implications, J. Assoc. for Comp. Mach., v.25, no.3, July 1978, pp.499-508.

[14] W. H. Gates and C. H. Papadimitriou, Bounds for sorting by prefix reversal, Discrete Mathematics, v.27, pp.47-57, 1979

[15] L. S. Heath and J. P. C. Vergara, Sorting by bounded block-moves, Discrete Applied Mathematics, v.88, i.1-3, p.181-206, 9 November 1998.

[16] C. Hurkens, L. Van Iersel, J. Keijsper, S. Kelk, L. Stougie and John Tromp, Prefix reversals on binary and ternary strings, Arxiv preprint math CO/0602456, 2006 - arxiv.org.

[17] T. Hartman and R. Shamir. A simpler and faster 1.5-approximation algorithm for sorting by transpositions. Information and Computation, Elsevier, 2005.

[18] M. Heydari and I. H. Sudborough, On the Diameter of the Pancake Network, J. of. Algorithms, v.25, pp.67-94, 1997.

[19] L. S. Heath and J. P. C. Vergara. Sorting by Short Swaps. Journal of Computational Biology, Vol. 10, No. 5, p. 775 -789, Oct 2003.

[20] J. Kececioglu and D. Sankoff, "Exact and Approximation Algorithms for Sorting by Reversals, with Application to Genome Rearrangement". Algorithmica 13, p.180-210, 1995.

[21] A. J. Radcliffe, A.D. Scott and E.L. Wilmer, Reversals and transpositions over finite alphabets, SIAM Journal on Discrete Math, v.19, no.1, pp.224-244, Aug 03 2005

[22] D. Shapira and J. A. Storer. Edit Distance with Move operations. Lecture Notes in Computer Science, Springer Berlin / Heidelberg, ISSN: 0302-9743, Volume 2373, p.85-98, 2002.