# SOME EXPERIMENTS ON THE SORTING BY REVERSALS PROBLEM

Lenwood S. Heath and John Paul C. Vergara

Department of Computer Science
Virginia Polytechnic Institute and State University
Blacksburg, Virginia 24061-0106
heath,jpv@cs.vt.edu

September 6, 1995

**Abstract**

*Sorting by reversals* is the problem of finding the minimum number of reversals required to sort a permutation $\pi$. The problem is significant with respect to the study of genome rearrangements and phylogeny reconstruction. This paper presents a programming framework for performing experiments on the problem. Several conjectures concerning optimal sorting sequences are tested using this framework.

## 1 Introduction

In molecular biology, two genomes are compared by analyzing their respective gene sequences. Phylogenetic trees of genomes are constructed based on information about how closely two genomes are related. A genomes evolves by insertions, deletions, duplications, reversals, and transpositions of its fragments. One may think of these mutations as operations applied to a string or permutation of genes. In studying the distance between two genomes, the biologist wants to determine the minimum number of operations required to transform one into the other.

Some algorithmic results on this problem have already been obtained. These results often involve restricting the operations on a genome to a subset of the operations identified above. Wagner [8] and Lowrance and Wagner [7] investigate *string-to-string correction*, a similar problem on strings involving edit operations such as insertion, deletion, and adjacent swaps of single symbols in the string. They show that the problem is NP-complete in general but solvable in polynomial time when only certain operations are allowed. Bafna and Pevzner [2] study *sorting by transpositions*. A transposition on a permutation of genes is defined as an operation that repositions a block of genes by inserting it elsewhere in the permutation. Notice that the problem can be restated as a sorting problem since the destination genome can be thought of as the sorted permutation of genes and one seeks the minimum number of transpositions required to sort a given genome. Bafna and Pevzner provide a metric on a permutation that uses a graph representation and derive upper and lower bounds and approximation algorithms for this problem. Guyer, Heath and Vergara [3] employ some heuristics based on subsequences and devise corresponding algorithms for the problem. They observe that selecting transpositions that maximally lengthen the longest increasing subsequence in the permutation appear to produce near-optimal results.

Given a permutation $\pi = \pi_1\pi_2\ldots\pi_n$, a *reversal* $\rho(i,j)$, where $1 \leq i < j \leq n$, has the effect of inverting the subsequence $\pi_i\pi_{i+1}\ldots\pi_j$ in $\pi$. More precisely, $\pi \cdot \rho(i,j) = \pi_1\pi_2\ldots\pi_{i-1}\pi_j\pi_{j-1}\ldots\pi_i\pi_{j+1}\pi_{j+2}\ldots\pi_n$. Consider the permutation $\pi = 256745981$, for example. Applying $\rho(4,8)$ to $\pi$ transforms $\pi$ to $\pi \cdot \rho = 256895471$. The *reversal distance* of a permutation $\pi$ is the minimum number of reversals whose product is $\pi$, equivalently, the minimum number of reversals required to sort $\pi$. In this paper, we investigate *sorting by reversals*, the problem of determining the reversal distance of a permutation.

Kececioglu and Sankoff [5, 6] and Bafna and Pevzner [1] introduce the notion of *breakpoints* in a permutation and devise approximation algorithms for sorting by reversals based on breakpoints. Bafna and Pevzner, in addition, identify upper and lower bounds for the reversal distance of a permutation. It is not yet known whether sorting by reversals is NP-complete or has a polynomial time algorithm. Hannenhalli and Pevzner [4], however, show that a variant of the problem, sorting *signed* permutations by reversals, is solvable in $O(n^4)$ time.

This paper reports on some of our ongoing research into sorting problems in computational biology. Here, we use the dynamic programming paradigm to devise and implement an algorithm that solves the problem for *all* permutations of length $n$. Of course, the algorithm runs in exponential time, which makes it practical only for small values of $n$, in particular for $n \leq 10$. However, the program is nevertheless useful because it allows us to test conjectures about a solution for an instance of the problem. For example, one could conjecture that there always exists a sequence of reversals that sorts $\pi$ optimally such that for each reversal $\rho(i, j)$, $\pi_i > \pi_j$ (here, $\pi$ represents an intermediate configuration of the permutation during the sort). The conjecture implies that a reversal should always correct the relative order of its end elements. To test this conjecture, we revise our dynamic programming algorithm and check for reversals of the specified type. The algorithm could then test whether optimal solutions for all size-$n$ permutations are obtained using this type of reversals.

Either we verify the conjecture for permutations of size $\leq 10$, or we obtain a counterexample that can help guide future directions. Counterexamples would then be produced if the conjecture is incorrect. If no counterexamples are produced, then we have discovered some fact about the problem that is true at least for permutations of size $\leq 10$.

The paper is structured as follows. Section 2 discusses the programming framework we use in our experiments. Section 3 enumerates the conjectures we test and the counterexamples that result. We summarize our work in Section 4.

## 2 The Programming Framework

An algorithm for determining the reversal distance for all permutations of size $n$ that runs in $O(n^3 n!)$ time is presented in Figure 1. It is called SortByRevs and takes as input an integer $n$.

The algorithm uses an array $P$ of $n!$ elements each with two fields, *dist* and *link*. Each permutation $\pi$ of size $n$ corresponds to an element in the array. We assume that $P$ is indexed by the set of size-$n$ permutations (in our actual implementation, we order

SortByRevs($n$) Sorting by reversals for all permutations of size $n$.

INPUT: An integer $n$.
OUTPUT: A *reversal tree* of all size-$n$ permutations rooted at the identity permutation, such that the length of the path from any node $p$ to the root represents the reversal distance of $p$.

```
▷ P[n!] is an array of {dist and link} as described in the text
   P is indexed by the set of all permutations of size n

▷ initialize P array
foreach permutation π of size n do
     begin
          P[π].dist ← UNDEFINED
          P[π].link ← NIL
     end
P[identity].dist ← 0
for k ← 0 to n − 2 do
     foreach permutation π where P[π].dist = k do
          foreach reversal ρ do
               ▷ determine all permutations accessible from π
                   using one reversal
               begin
                    σ ← π · ρ
                    if P[σ].dist = UNDEFINED then
                         begin
                              P[σ].dist ← k + 1
                              P[σ].link ← π
                         end
               end
return P
```

Figure 1: Algorithm SortByRevs.

the permutations in lexicographic order, map the permutations to integers, and then use integers as indices to the array).

Once algorithm SortByRevs executes, the *dist* field of $P[\pi]$ (for all permutations $\pi$ of size $n$) will contain the reversal distance of $\pi$. In addition, a *reversal tree* will result where the identity permutation is the root node and the *link* field of a node $P[\pi]$ is a pointer to its parent node. The unique path from node $P[\pi]$ to the root represents a transformation of $\pi$ to the identity permutation using the minimum number of reversals. $P[\pi].dist$ is exactly the length of this path.

The algorithm sets the distances and links of the nodes as follows. First, $P[identity].dist$ is set to 0. Then, the program iteratively sets to $k+1$ all the *dist* fields of all unvisited nodes accessible from nodes whose *dist* field $= k$. Let $P[\sigma]$ be a node and let $P[\pi]$ be a node with $P[\pi].dist = k$ and $\sigma = \pi \cdot \rho$ for some reversal $\rho$. Then, $P[\sigma].dist$ is set to $k+1$ and $P[\sigma].link$ is set to $\pi$.

Bafna and Pevzner [1] have shown that the reversal distance of a permutation of size $n$ is at most $n-1$. This means the program needs to iterate from $k=0$ up to $k=n-2$.

Note that the transformation represented by the path from a node to the root is not typically the only way to optimally sort $\pi$. The program determines only one such way to sort. This is because $P[\sigma].dist$ is set once a permutation $\pi$ (with distance $k$) and a reversal $\rho$ has been encountered with $\sigma = \pi \cdot \rho$. The distance is not reset even when the same condition applies later in the execution.

Algorithm SortByRevs runs in $O(n^3 n!)$ because all permutations $\pi$ (there are $n!$ of these) are eventually visited and all permutations accessible from $\pi$ (there are $n^2$ of these, and it requires $O(n)$ to perform the reversal that obtains one such permutation) are tested for each $\pi$.

Algorithm SortByRevs2, shown in Figure 2, is a revised version of the algorithm. Two additional fields, *reps* and *mark* are added to each array element. The algorithm has two main enhancements. The first enhancement is that the program detects whether there are other ways to reach a node; i.e., for every permutation $\sigma$, it detects whether there is

SortByRevs2($n$) Sorting by reversals for all permutations of size $n$.

INPUT: An integer $n$.
OUTPUT: A *reversal tree* of all size-$n$ permutations

```
▷ P[n!] is an array of {dist, link, reps, mark}
foreach permutation π of size n do
    begin
        P[π].dist ← UNDEFINED
        P[π].link ← NIL
        P[π].reps ← 0
        P[π].mark ← FALSE
    end
P[identity].dist ← 0
P[identity].mark ← TRUE
for k ← 0 to n − 2 do
    foreach permutation π where P[π].dist = k do
        foreach reversal ρ do
            begin
                σ ← π · ρ
                if P[σ].dist = k + 1 then
                    P[σ].reps ← P[σ].reps + 1
                else if P[σ].dist = UNDEFINED then
                    begin
                        P[σ].dist ← k + 1
                        P[σ].link ← π
                    end
                if P[σ].dist = k + 1 then
                    if P[σ].mark = FALSE and Check(σ, ρ) then
                        ▷ Check is a routine that tests if a reversal
                          is of an acceptable type
                        begin
                            P[σ].mark ← TRUE
                            P[σ].link ← π
                        end
            end
return P
```

Figure 2: Algorithm SortByRevs2.

more than one permutation $\pi$ with a reversal $\rho$ satisfying $P[\pi].dist = k$ and $\sigma = \pi \cdot \rho$. When this occurs, a field $P[\pi].reps$ is incremented. The other enhancement is that a conjecture on how reversals are chosen can be tested. A CHECK routine on $\sigma$ and $\rho$ is incorporated that tests whether a reversal on a permutation is acceptable. Operations that pass through the CHECK routine are "preferred" over other permutations and operations. The field $P[\pi].mark$ is set to TRUE when a link that passes the CHECK constraint occurs. Effectively, all those nodes whose *mark* fields still have FALSE values upon completion are counterexamples to the conjecture. As an example, consider the conjecture in Section 1. The check routine would be as follows:

CHECK$(\pi, \rho(a, b))$ $\quad\quad$ $\{$**return** $\pi_a > \pi_b\}$.

## 3 Conjectures and Counterexamples

In this section, we enumerate some of the conjectures we have tested using the program described in Section 2. The three conjectures described here are all incorrect and the program has generated counterexamples for each of them. Although these do not directly help in a solution to the problem, the counterexamples provide some insight into why this problem is hard and aid in the search for better solutions.

The first conjecture we mention suggests an algorithm that, on every step, decides between two reversals.

**Conjecture 1** *There exists a sequence of reversals that optimally sorts a permutation $\pi$ such that each reversal positions either the minimum or the maximum unpositioned element.*

For example, the permutation $\pi = 34251$ is optimally sorted by the following sequence of operations: $34251 \rightarrow 34215 \rightarrow 12435 \rightarrow 12345$. Notice that the three reversals that sort this permutation positions 5, 1, and 3 (or 4) respectively. In each case, the element is either a minimum or maximum unpositioned element. The first counterexample our program generates is $\pi = 25314$, a permutation optimally sorted by the following sequence:

**Counterexample:** $25314 \rightarrow 21354 \rightarrow 21345 \rightarrow 12345.$

Notice that the first reversal positions neither 1 nor 5 and attempting to do so requires more than 3 reversals to sort the permutation. The conjecture we introduced in Section 1 generalizes the previous conjecture.

**Conjecture 2** *There exists a sequence of reversals that optimally sorts a permutation $\pi$ such that each reversal places its end elements in their correct relative order.*

For $n \leq 5$, no counterexamples exist for this conjecture. However, the following counterexample results for $n = 6$.

**Counterexample:** $453162 \rightarrow 456132 \rightarrow 165432 \rightarrow 123456.$

The first reversal involves elements 316 and the reversal destroys the relative order of elements 3 and 6. It can be verified that this is the only way to proceed if the permutation is to be sorted optimally.

The last conjecture we state here involves breakpoints. A breakpoints exists between $\pi_i$ and $\pi_{i+1}$ if $|\pi_i - \pi_{i+1}| \neq 1$. A breakpoint exists before $\pi_1$ if $\pi_1 \neq 1$; likewise, a breakpoint exists after $\pi_n$ if $\pi_n \neq n$. Clearly, when $\pi$ is sorted, the number of breakpoints in $\pi$ is 0 so that a goal when sorting is to decrease the number of breakpoints in a permutation. Kececioglu and Sankoff [5, 6] and Bafna and Pevzner [1] study breakpoints and discuss an approximation algorithm that greedily selects a reversal that causes the maximum net decrease in number of breakpoints. We use our program to determine when the heuristic fails.

**Conjecture 3** *There exists a sequence of reversals that optimally sorts a permutation $\pi$ such that each reversal causes the maximum net decrease in number of breakpoints in $\pi$.*

**Counterexample:** $416523 \to 146523 \to 143256 \to 123456$.

Notice that the reversal involving 165 removes two breakpoints while the chosen reversal (41) removes only one. However, the 165 reversal does not lead to an optimal sequence of reversals. No counterexamples exist for $n < 6$.

# 4 Conclusions

We have devised an effective programming framework that allows us to test conjectures for the sorting by reversals problem. The three conjectures presented here and the counterexamples generated by the program demonstrate how difficult this problem is.

We can use the same technique to analyze restricted versions of the problem as well as related problems since this will entail only a change in the operations applied to the permutations.

# References

[1] V. Bafna and P. A. Pevzner, *Genome rearrangements and sorting by reversals*, in Proceedings of the 34th Annual IEEE Symposium on Foundations of Computer Science, 1993, pp. 148–157.

[2] ——, *Sorting permutations by transpositions*, in Proceedings of the Sixth Annual ACM-SIAM Symposium on Discrete Algorithms, 1995, pp. 614–623.

[3] S. Guyer, L. S. Heath, and J. P. C. Vergara, *Subsequence and run heuristics for the sorting by transpositions problem.* Accepted to the Fourth DIMACS International Algorithm Implementation Challenge, 1995.

[4] S. Hannenhalli and P. Pevzner, *Transforming cabbage into turnip (polynomial algorithm for sorting signed permutations by reversals)*, in Twenty-Seventh Annual ACM Symposium on the Theory of Computing, 1995, pp. 178–189.

[5] J. Kececioglu and D. Sankoff, *Exact and approximation algorithms for the inversion distance between two chromosomes*, in Fourth Annual Symposium on Combinatorial Pattern Matching, Lecture Notes in Computer Science, vol. 684, 1993, pp. 87–105.

[6] ——, *Efficient bounds for oriented chromosome inversion distance*, in Fifth Annual Symposium on Combinatorial Pattern Matching, Lecture Notes in Computer Science, 1994, pp. 307–325.

[7] R. Lowrance and R. A. Wagner, *An extension of the string-to-string correction problem*, Journal of the ACM, 22 (1975), pp. 177–183.

[8] R. A. Wagner, *On the complexity of the extended string-to-string correction problem*, in Proceedings of the 7th Annual ACM Symposium on Theory of Computing, 1975, pp. 218–222.