Sorting by reversals

Bogdan Paşaniuc *

May 2, 2005

1 Introduction

One of the most studied problems in the field of computational biology is the string matching problem. Much of the reasearch has focused on developing efficient algorithms for transforming one string into another by minimizing the number of steps. The biological motivation for this problem comes from the fact that DNA sequences get transformed by a series of basic operations such as mutations, deletions and substitutions.

When trying to understand how genetic sequences mutate at the chromosome level we need to consider more global operations, rather than the basic ones. One of the most common global mutation is the reversing of a substring, operation also known as a *reversal*. In fact, the minimum number of reversals used to transform one string into another has proven to be a very good estimate for the evolutionary distance between organisms.

The problem presented in this paper is finding the minimum number of reversals needed to transform one string into another. Two variants of this problem together with results taken from [1] and [2] are going to be presented in the following sections.

2 Definition of the problem

In order to represent the sequence of genes on a chromosome, we are going to use permutations. More clearly, we are given the order of n genes in two related organisms; we only consider the genes that appear in both organisms. The order of the genes are represented by a permutation $\sigma = (\sigma_1, \sigma_2, \cdots, \sigma_n)$, where by σ_i we denote the position where the gene i appears. A reversal of the interval $[i, j]$ is a permutation

$$\rho = \begin{pmatrix} i & i+1 & \cdots & j \\ j & j-1 & \cdots & i \end{pmatrix},$$

*Computer Science and Engineering Department, University of Connecticut, 371 Fairfield Rd., Unit 2155, Storrs, CT 06269-2155. E-mail: bogdan@engr.uconn.edu.

with $\rho(k) = k$ for $k \notin [i, j]$. The effect of reversing the order of a subsequence of genes can be simulated by the composition of permutations; $\sigma \circ \rho$ is a new permutation in which the order of the genes $\sigma_i, \sigma_{i+1}, \cdots, \sigma_j$ has been reversed.

Definition 1 *The **reversal distance problem** on permutations is, given permutations σ and γ, find the minimum number d of reversals $\rho_1, \rho_2, \cdots, \rho_d$, such that*

$$\sigma \circ \rho_1 \sigma \circ \rho_2 \cdots \sigma \circ \rho_d = \gamma.$$

We call d the *reversal distance* between γ and σ. Because of the fact that the reversal distance between σ and γ is equal with the distance between $\gamma^{-1} \circ \sigma$ and ι (the identity permutation) the problem of computing the reversal distance between two permutations is equivalent to the problem of computing the reversal distance between a permutation and the identity permutation. This new formulation of the problem is known as *sorting by reversals*.

Depending on how a reversal transforms a permutation, we can distingush between two variants of the problem. If, whenever we perform a reversal, the sign of the reversed values is also changed than we call the new formulation *sorting signed permutations by reversals* [3]; if the sign remains unchanged, new problem is called *sorting permutations by unsigned reversals* [1].

Although the two problems are closely related, the unsigned version was proven to be NP-complete [4], while a polynomial algorithm for the signed version was presented in [3].

3 Sorting unsigned permutations

In this section we are going to present a 2-approximation algorithm and an exact branch and bound algorithm [1] for sorting unsigned permutations. Although the approximation factor has been improved to 3/2 [5], the algorithm presented below is the first constant factor algorithm to be proposed for the problem.

3.1 An approximation algorithm

The most natural algorithm for sorting by reversals is to bring element 1 into place, followed by the second element and so on, up to element n. Formally at step i perform the reversal $[i, \sigma_i^{-1}]$ if $\sigma_i \neq i$. By bringing the $n-1$ element into position at step $n-1$, we also bring the n element into position n. For each permutation of size n, this simple algorithm sorts the permutation using at most $n-1$ reversals.

This algorithm can perform arbitrarily poorly. For example, if we consider the permutation $\sigma = (n, 1, 2, \cdots, n-1)$, then this algorithm will bring 1 into place and so on until $n-1$, thus using $n-1$ reversals. Considering that the given permutation can be solved by 2 reversals, namely $[2, n]$, followed by $[1, n]$, then the solution of the given algorithm is $1/2(n-1)$ times away from the optimum.

One of the most important results of [1] is the first constant factor approximation algorithm for sorting unsigned permutations by reversals. We are going to present this algorithm next, but first we need to give some definitions.

Definition 2 *A **breakpoint** of a permutation σ is a pair of adjacent positions $[i, i+1]$ such that σ_i and σ_{i+1} are consecutively decreasing or increasing. A **strip** of σ is a maximal run of consecutively increasing(decreasing) elements.*

A very important observation is that a reversal $[i, j]$ affects the breakpoints in σ only at it's end positions, namely $[i-1, i]$ and $[j, j+1]$. The pairs $[k, k+1]$ outside the reversal are left unchanged, while the pairs inside the reversal are changed from decreasing to increasing or from increasing to decreasing.

If we denote by $\Phi(\sigma)$ the number of breakpoints in σ, then the number of breakpoints removed by a reversal ρ is

$$\Delta\Phi(\sigma, \rho) = \Phi(\sigma) - \Phi(\sigma \circ \rho).$$

Since a reversal affects only 2 breakpoints, $\Delta\Phi(\sigma)$ can take values between -2 (introduces 2 breakpoints) and 2 (removes 2 breakpoints).

In order to handle the boundaries for any $\sigma = (\sigma_1, \cdots, \sigma_n)$ we introduce two new elements $\sigma_0 = 0$ and $\sigma_{n+1} = n + 1$. We "fix" these two new elements, that is, for any reversal $[i, j]$, $1 \leq i, j \leq n$. It follows that $(0, 1)$, respectively $(n, n+1)$ are breakpoints if $\sigma_1 \neq 1$, respectively $\sigma_n \neq n$.

The identity permutation has no breakpoints while any other permutation has some breakpoints. A greedy algorithm comes naturally, as at each step we try to remove as many breakpoints as possible.

Greedy Algorithm

while (σ contains a breakpoint) **do**

1. find a reversal ρ_i such that $\Delta\Phi(\sigma, \rho)$ is maximized. If $\Delta\Phi(\sigma, \rho) = 1$ then favor reversals that leave a decreasing strip.
2. $\sigma = \sigma \circ \rho_i$
3. $i = i + 1$

endwhile

We will next prove that the algorithm above is a 2-approximation algorithm. The basic idea of the proof is that after k reversals chosen by the algorithm the number of breakpoints has decreased with at least k. We will show the claim above using following three facts.

Lemma 1 *Every permutation with a decreasing strip has a reversal that removes a breakpoint.*

Proof. Let $\rho(i', i)$ be the decreasing strip such that π_i is the smallest. It follows that the element $\pi_i - 1$ must be in an increasing strip that is situated to the left or to the right of the decreasing strip. As we can see from **Figure 1**, in either of the two cases there is a revesal that removes one breakpoint.

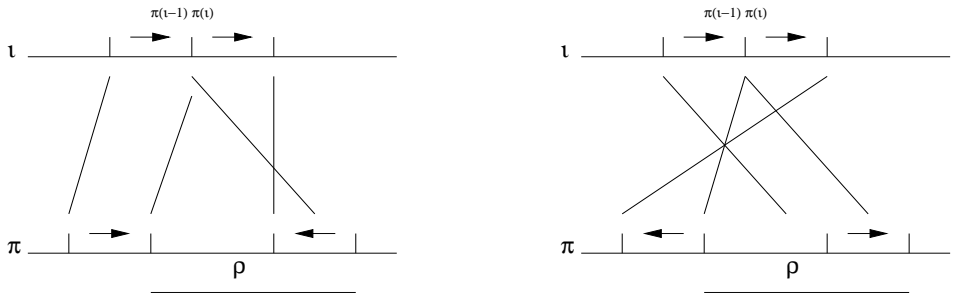

Figure 1: A permutation π with a decreasing strip has a reversal ρ that removes a breakpoint. π_i is the smallest element that lies in a decreasing strip.

Lemma 2 *If every reversal that removes a breakpoint leaves a permutation with no decreasing strip then there exists a reversal that removes two breakpoints.*

Proof. Consider the decreasing strip containing the smallest element π_i, than the element $\pi_i - 1$ must be in an increasing strip. Following **Figure 1** notice that if the increasing strip lies to the right then the considered reversal leaves a decreasing strip. Because every reversal that removes a breakpoint leaves no decreasing strips, the increasing strip must lie to the left. Denote by ρ_i the reversal that removes a breakpoint, as in previous lemma.

If we consider the decreasing strip whose first element π_j is the largest, then element $\pi_j + 1$ must lie in an increasing strip. In a similar manner as before, one proves that the increasing strip must lie to the right of the decreasing strip containing π_j. Let ρ_j be the reversal that removes one breakpoint.

By using the fact that the reversal leaves a permutation with no decreasing strip, one can easily prove that the two reversals considered above (ρ_i and rho_j) are one and the same, thus proving the existance of a reversal that removes two breakpoints.

Using these two lemmas, we can prove by induction on the number of breakpoints in a permutation, that the following result holds.

Lemma 3 *The above algorithm sorts a permutation σ with a decreasing strip in at most $\Phi(\sigma) - 1$ reversals.*

Then it comes clearly that the algorithm presented above sorts every permutation in at most $\Phi(\sigma)$ reversals. If a permutation has a decreasing strip then we can apply the previous lemma. If a permutation has no decreasing strip then any reversal creates a decreasing strip. In other words after one step we can apply the previous lemma again giving a worst case of $1 + (\Phi(\sigma) - 1) = \Phi(\sigma)$.

Theorem 1 *The above algorithm is a 2-approximation algorithm for sorting by reversals.*

Proof Let us denote by $OPT(\sigma)$ the minimum number of reversals required by greedy. Since a reversal can remove at most two breakpoints then $OPT \geq \Phi(\sigma)$. But $Greedy(\sigma) \leq \Phi\sigma)$. We conclude that $OPT(\sigma) \geq Greedy(\sigma)/2$.

3.2 An exact branch and bound algorithm

The branch and bound algorithm simply searches through all possible solutions in order to find the minimum number of reversals needed. In order to prune as much as possible from the search tree, we need tight lower and upper bounds.

A simple lower bound on $Opt(\sigma)$ is $\Phi(\sigma)$, but this bound is extremely weak because it assumes every breakpoint of σ can be eliminated by a reversal that removes 2 breakpoints, which in practice is rarely achieved.

Mainly we need to answer the following question: how many breakpoints can be removed by reversals that remove two breakpoints?

We can efficiently construct a graph where every vertex is a breakpoint and there is an edge between two breakpoints if these two breakpoints can be eliminated by a single reversal. Now we can say that the larget number of reversals that remove two breakpoints we can perform is the size of the maximum cardinality matching.

We denote by m the number of vertices covered by a maximum cardinality matching, namely twice the size of the matching. We need to remove another $\Phi(\sigma) - m$ breakpoints. The best scenario is to perform a reversal that removes one breakpoint followed by a reversal that removes two breakpoints.

Notice that a reversal that removes only one breakpoint, can affect only one additional breakpoint; in other words in order to perform a reversal that removes two breakpoints we need to perform one that removes one breakpoint. This implies that the best that we can do is to remove 3 breakpoints at the price of 2 reversals.

If we compute the total number of reversals used we obtain a new lower bound of:

$$\lceil 1/2m + 2/3(\Phi(\sigma) - m) \rceil$$

We can further improve this bound by using t-uples of breakpoints and computing the minimum number of reversals needed to remove t breakpoints.

As the upper bound one can use the Greedy algorithm presented below.

4 Sorting signed permutations

In the signed version of the problem, each element of the permutation has a sign; a reversal $\rho(i, j)$ transforms a permutation $\pi = (\pi_1 \cdots \pi_n)$ into

$$\pi' = (\pi_1, \cdots, \pi_{i-1}, -\pi_j, \cdots, -\pi_i, \pi_{j+1}, \cdots, \pi_n).$$

This variant of the problem comes from the biological fact that a gene has an orientation on the DNA strand and, by performing a reversal, the orientation is changed.

In [3] the first polynomial algorithm for sorting signed permutation was presented. The main problem is in choosing a *safe* reversal. How this safe reversal is chosen depends on the construction of a permutation of size $2n$ associated to the original permutation and analyzing the cycles and the connected components of graphs associated with this permutation.

In [2] the authors present an elementary treatment of the sorting of the *oriented components* of a permutation, together with a simplified definition of the concept of a *hurdle*, thus giving the first completely elementary algorithm. In the following section the improvements made to the original algorithm of Hannenhalli and Pevzner are sumarized.

4.1 Basic Algorithm

An *oriented pair* (π_i, π_j) is a pair of consecutive integers such that $|\pi_1| - |\pi_j| = \pm 1$ and $sign(\pi_i) \neq sign(\pi_j)$. An oriented pair indicates a reversal that creates consecutive elements in the permutation. The reversal induced by an oriented pair (π_i, π_j) is

$$\rho = \begin{cases} \rho(i, j-1) & if \quad \pi_i + \pi_j = +1 \\ \rho(i+1, j) & if \quad \pi_i + \pi_j = -1 \end{cases}$$

For example, the permutation (0 1 5 4 -2 3 6) has two oriented pairs, namely (1 -2) and (-2 3). The reversal induced by (1 -2) is

$$(0\ 1\ \underline{5\quad 4\ -2}\ 3\ 6\ )$$

transforming π into

$$(0\ 1\ 2\ -4\ -5\ 3\ 6\ )$$

and thus creating the pair of consecutive integers (1 2).

Reversals that create consecutive integers are always induced by oriented pairs. When a reversal creates a consecutive pair of integers then it is called *oriented reversal*. The number of oriented pairs, an oriented reversal introduces is called the *score of the reversal*. A simple greedy strategy would be to choose at each step an oriented reversal that introduces the maximum number of oriented pairs.

Algorithm 1: As long as π has an oriented pair, choose the oriented reversal that has the maximal score.

After applying this algorithm we always obtain a permutation with positive elements.

If we assume that there is a negative element then there exists at least one oriented pair. That contradicts with the stopping condition of the algorithm.

Claim 1: If the previous algorithm applies k reversals to π, yielding π', then $d(\pi) = d(\pi') + k$, where $d(\pi)$ is the minimum number of reversals to transform π into the identity permutation.

Most reversals applied to a permutation with positive elements will create oriented pairs. In the next section we discuss how to choose the optimal one.

4.2 Sorting positive permutations

A positive permutation π is *reduced* if π does not contain consecutive elements. As previously π is framed by 0 and $n+1$. We consider the circular order induced by setting 0 to be the succesor of $n+1$.

A *framed interval* in π is an interval:

$$i, \pi_{i+1}, \pi_{i+2}, \cdots, \pi_{i+k-1}, i+k$$

such that all integers between i and $i+k$ belong to the interval $[i, i+k]$.

If π is reduced, a *hurdle* of π is a framed interval that contains no other framed interval. We introduce two operations on hurdles with respect to reversals.

Hurdle cutting consists of performing a reversal between the elements i and $i+1$

$$i, \underline{\pi_{j+1}, \pi_{j+2}, \cdots}, i+1, \cdots \pi_{i+k-1}, i+k,$$

where $[i, i+k]$ is a hurdle.

Hurdle merging consists of performing a reversal on the endpoints of two hurdles

$$i, \cdots, \underline{i+k, \cdots i'}, \cdots i'+k',$$

with $[i, i+k]$ and $[i', i'+k']$ being hurdles.

A *simple hurdle* is a hurdle whose cutting decreases the number of hurdles. Hurdels that are not simple are called *super hurdles*.

Algorithm 2: If a permutation has $2k$ hurdles then merge any two non-consecutive hurdles. If the number of hurdles is $2k+1$, then if it has one simple hurdle then cut it; if it has no simple hurdles then merge two non-consecutive ones.

By combining the two algorithms one can optimally sort any signed permutation.

4.3 Algorithm soundness

The claims made in the preceeding sections will be proved by using some elements from the Hannenhalli-Pevzner theory.

First we construct the *breakpoint graph*. For a given permutation π, each positive element x is replaced by the sequence $2x-1$ and $2x$ and each negative element$(-x)$ by $2x$ and $2x-1$. For example

$$\pi = (0, -1, 3, 5, 4, 6, -2, 7)$$

becomes

$$\pi' = (0, 2, 1, 5, 6, 9, 10, 7, 8, 11, 12, 4, 3, 13)$$

Based on π' we construct the breakpoint graph in the following manner. Straight edges join every other pair of consecutive elements, and curved edges, called *arcs*, join every other pair of consecutive integers, starting with $(0, 1)$

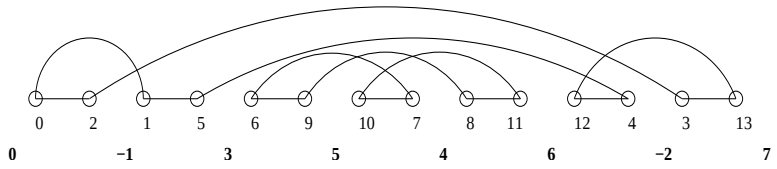

Figure 2: Breakpoint graph associated to $\pi = (0, -1, 3, 5, 4, 6, -2, 7)$

Starting with the breakpoint graph we can construct the *arc overlap graph*, graph whose edges are the arcs of the previous graph and there is an edge between the nodes if the arcs overlap in the breakpoint graph.

The *support* of an arc is the interval of elements of π' between, and including it's endpoints.

An arc is *oriented* if its support contains an odd number of elements, otherwise it is *unoriented*. Oriented vertices in the arc overlap graph correspond to oriented pairs in the original permutation. Oriented vertices are marked by black dots (*Figure 3*).

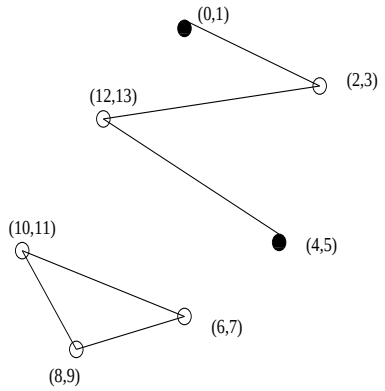

Figure 3: Arc overlap graph associated to $\pi = (0, -1, 3, 5, 4, 6, -2, 7)$

If we define an *oriented component* in the arc overlap graph as a connected component that contains an oriented vertex, then a *safe* reversal is a reversal that does not create any new unoriented components.

Theorem 2 ***(Hannenhalli and Pevzner, 1995).*** *Any sequence of oriented safe reversals is optimal.*

The difficulties in sorting oriented components lie in the detection of safe reversals, but by using some of the properties of the arc overlap graph one can prove that an oriented reversal of maximal score is safe. It follows that the elementary strategy of choosing the reversal with maximal score is optimal.

The concept of a hurdle, as defined in the section before, is equivalent to a minimal unoriented component in the arc overlap graph.

5 Conclusion

The problem of sorting by reversals has been the studied by many researchers, and from a problem of unknown complexity [1] it became a NP-Hard problem [4], while an interesting variant of it was proven to be polynomial [3]. The first polynomial algorithm for the signed version, proposed by Hannenhalli and Pevzner in 1999 relies on the analysis of intermediary constructions. The most important contribution of [2] is that it gives the first completely elementary solution to the signed version of the problem.

References

[1] Kececioglu, J. and Sankoff, David. Exact and approximation algorithms for sorting by reversals, with application to genome rearrangement. *Algorithmica*, 13(1/2):180-210, 1995.

[2] Bergeron, Anne. A Very Elementary Presentation of the Hannenhalli-Pevzner Theory. *Discrete Applied Mathematics*, 146(2):134-145, 2005.

[3] Hannenhalli, S. and Pevzner, Pavel A.. Transforming cabbage into turnip: polynomial algorithm for sorting signed permutations by reversals. *Journal of the ACM*, 46(1):1-27, 1999.

[4] Caprara, Alberto. Sorting by reversals is difficult. *RECOMB 1997*, ACM Press: 75-83, 1997.

[5] Christie, D. A. A 3/2-approximation algorithm for sorting by reversals. *In Proc. ninth annual ACM-SIAM Symp. on Discrete Algorithms (SODA 98)*, ACM Press: 244-252, 1998.