CS310 - Independent Study Report

Deepak Garg - 1999144
Akash Lal - 1999131
13 Algorithms

14 Algorithm given by [KST]
   14.1 Overview of the Algorithm .......................... 36
   14.2 Finding a Happy Clique ............................ 36
   14.3 Searching the Happy Clique ........................ 39
   14.4 Algorithm and Running Time ....................... 39

15 Algorithm given by [ANB] ................................. 40

16 Algorithm given by Us .................................... 41
   16.1 The Algorithm ..................................... 41
   16.2 An Unsolved Problem ............................... 43

17 Acknowledgement ........................................... 44

18 Bibliography ................................................ 44

19 Appendix ..................................................... 46

20 Diagrams ..................................................... 48
1 Definition and Motivation

A genome of $n$ genes can be described as a permutation $\pi = (\pi_1, \ldots, \pi_n)$ on the integers $1, \ldots, n$. In a signed permutation, each element of the permutation has an additional sign. A reversal $\rho(i, j)$ acting on a permutation $\pi$ transforms it into a permutation $\pi'$ such that,

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

Reversal distance is defined as the minimum number of reversals needed to transform a permutation to the identity permutation $i = (+1, +2, \ldots, +n)$ and the sorting by reversal problem is to find this sequence of reversals. This problem can be equivalently defined as finding the minimum sequence of reversals that will transform a given permutation $\pi$ to another given permutation $\phi$. These problems are equivalent since any permutation $\pi$ can also be described by a function that maps $k$ to $\pi_k$ and vice versa. Using this we can easily see that the reversals that will transform $\pi$ to $\phi$ will also transform $\phi^{-1}\pi$ to $i$ where the inverse and product of permutations is taken on their corresponding functions. This view of the problem is what has attached importance to it.

With the current explosion in genomic information the need for comparing genomes has also increased. Various metrics have been proposed to measure distance between two genomes. Most of these are based on some operations that can transform a genome into another one and defining the distance as the minimum number of operations required. One such common measure is the edit distance which allows for insertion, deletions and modification of a single gene. Such operations, which have local effect, are called local operations. Other operations such as reversals have a global effect and are called global operations. People have realised that global operations might be more com-
mon in nature than local operations and the distance measure based on it might be more realistic. Thus, sorting by reversal has gained importance.

2 Common Terminology

As has been common practice in this field, we will transform each signed permutation into an unsigned one by replacing each positive element $x$ by $2x − 1$, $2x$ and each negative element $−x$ by $2x$, $2x − 1$ thus obtaining a permutation on double the number of elements. We then augment each such transformed permutation $\pi = (\pi_1, \ldots, \pi_n)$ by two integers, $\pi_0 = 0$ and $\pi_{n+1} = n + 1$. This transformation is a one-to-one correspondence if we keep the pairs $2x, 2x − 1$ together across reversals. Now, if $\pi$ is the transformed permutation of $\phi$ then the reversal $\rho(2i + 1, 2j)$ on $\pi$ will mimic the reversal $\rho(i, j)$ on $\phi$. Therefore, we will only allow reversals of the form $\rho(2i + 1, 2j)$. From now on, all permutations will be transformed permutations and all reversals will be restricted as told.

**Breakpoint** : A pair $(\pi_i, \pi_{i+1})$ is called a **breakpoint** if $|\pi_i − \pi_{i+1}| \neq 1$.

**Adjacency** : A pair $(\pi_i, \pi_{i+1})$ is called an **adjacency** if it is not a breakpoint i.e. $|\pi_i − \pi_{i+1}| = 1$.

**Reversal** : A reversal $\rho(i, j)$ will transform a permutation $\pi = (\pi_1, \ldots, \pi_n)$ to,

$\pi' = \pi \rho(i, j) = (\pi_1, \ldots, \pi_{i-1}, -\pi_j, -\pi_{j-1}, \ldots, -\pi_{i}, \pi_{j+1}, \ldots, \pi_n)$

We say that the reversal acts on the pairs $(\pi_{i-1}, \pi_i)$ and $(\pi_j, \pi_{j+1})$. 
**Breakpoint Graph**: The breakpoint graph \( B(\pi) = (V, E_b, E_g) \) of a permutation \( \pi = (\pi_0, \ldots, \pi_{n+1}) \) is an edge-colored graph with \( E_b \) being black edges and \( E_g \) being gray edges.

\[
V = \{\pi_0, \ldots, \pi_{n+1}\} = \{0, \ldots, n + 1\}
\]
\[
E_b = \{(\pi_i, \pi_j) | j = i + 1 \text{ and } (\pi_i, \pi_{i+1}) \text{ is a breakpoint }\}
\]
\[
E_g = \{(\pi_i, \pi_j) | j \neq i + 1 \text{ and } |\pi_i - \pi_j| = 1\}
\]

The above definitions are illustrated in figure 1a. The breakpoint graph has certain important properties.

**Property 2.1**: Each vertex has degree 0 or 2.

**Property 2.2**: Each vertex with degree 2 has one black edge and one gray edge incident to it.

To see why these properties hold, first note that due to the transformation we have applied to convert signed permutations into unsigned ones, for each element \( \pi_i \) (\( i \neq 0 \) or \( n + 1 \)) either \( \pi_i + 1 \) or \( \pi_i - 1 \) neighbours it. Since black edges can only be between consecutive elements of a permutation and that too when their difference in not one, there can be atmost one black edge incident on a vertex. Similarly, gray edges can only be between numbers differing by one but not placed consecutively, there can be atmost one gray edge incident on a vertex. Now all that remains to be shown is that if one gray edge is incident to a vertex then one black edge will be incident on it as well and vice versa. This happens because a gray edge on \( \pi_i \) implies that one of the elements \( \pi_i + 1, \pi_i - 1 \) does not lie adjacent to \( \pi_i \). This means that one of the element adjacent to \( \pi_i \) differs with it by more than one. Hence a black edge will be incident on \( \pi_i \). The other direction can be proved similarly. The case for \( \pi_0 \) and \( \pi_{n+1} \) is a little different but can be proved easily.
These properties together imply that the graph $B(\pi)$ has a unique cycle decomposition and each cycle has edges whose color alternates as we go around the cycle. For this reason, the cycles are called *alternating cycles*.

**Notation 2.1:** Let $d(\pi)$ be the minimum number of reversals required to sort $\pi$. Also let $b(\pi)$ be the number of breakpoints (i.e. black edges) in $B(\pi)$ and $c(\pi)$ be the number of cycles in $B(\pi)$. The number $c(\pi)$ is well defined because of the above result.

**Notation 2.2:** Define $\Delta b(\pi, \rho) = b(\pi \rho) - b(\pi)$ and $\Delta c(\pi, \rho) = c(\pi \rho) - c(\pi)$. We will omit writing $\rho$ and $\pi$ inside $\Delta b(\pi, \rho)$ and $\Delta c(\pi, \rho)$ when they are clear from the context.

Now we are ready to prove some results.

## 3 Lower Bounds

The first thing we note is that any reversal cannot decrease the number of breakpoints by more than two. This is because any reversal changes only two pairs i.e. $\rho(i, j)$ destroys the pairs $(\pi_{i-1}, \pi_i)$ and $(\pi_j, \pi_{j+1})$ and creates the pairs $(\pi_{i-1}, \pi_j)$ and $(\pi_i, \pi_{j+1})$ thus reducing the number of breakpoints by at most two. Since the identity permutation has no breakpoints, this gives us the following lower bound,

$$d(\pi) \geq \frac{b(\pi)}{2}$$

This can be improved by doing some more case analysis on the reversals acting on a permutation. Bafna and Pevzner [5] gave the following results which show the various values attainable by $\Delta b$ and $\Delta c$. Here case analysis is done on the pairs on which
the reversal $\rho(i, j)$ is acting.

**Theorem 3.1 :** Depending on the two end points of a reversal, we have for

1. Two adjacencies: $\Delta c = 1$ and $\Delta b = 2$.
2. A breakpoint and an adjacency: $\Delta c = 0$ and $\Delta b = 1$.
3. Two breakpoints each belonging to a different cycle: $\Delta c = -1$ and $\Delta b = 0$.
4. Two breakpoints of the same cycle $C$:
   (a) $(\pi_i, \pi_{j+1})$ and $(\pi_{i-1}, \pi_j)$ are gray edges: $\Delta c = -1$ and $\Delta b = -2$.
   (b) Exactly one of $(\pi_i, \pi_{j+1})$ and $(\pi_{i-1}, \pi_j)$ is a gray edge: $\Delta c = 0$ and $\Delta b = -1$.
   (c) Neither $(\pi_i, \pi_{j+1})$ nor $(\pi_{i-1}, \pi_j)$ is a gray edge, and when breaking $C$ at $i$ and $j$ vertices, $i - 1$ and $j + 1$ end up in the same path: $\Delta c = 0$ and $\Delta b = 0$.
   (d) Neither $(\pi_i, \pi_{j+1})$ nor $(\pi_{i-1}, \pi_j)$ is a gray edge, and when breaking $C$ at $i$ and $j$ vertices, $i - 1$ and $j + 1$ end up in different paths: $\Delta c = 1$ and $\Delta b = 0$.

A proof for the above in give in the appendix since it uses notation which has not been introduced yet.

From the above cases we get that $\Delta b - \Delta c \geq -1$. Noting the following result, we can prove another lower bound in $d(\pi)$.

**Lemma 3.1 :** The identity permutation is the only permutation with $b(\pi) - c(\pi) = 0$.

**Proof :** A cycle must contain at least four edges because it is an alternating cycle as proved above. It must therefore contain at least two black edges. Thus, $c(\pi) \geq 2b(\pi)$. Now, $b(\pi) = c(\pi)$
implies that $b(\pi) = 0$ which is only true for the identity permutation.

This gives us the following result.

**Theorem 3.2:**

$$d(\pi) \geq b(\pi) - c(\pi)$$

## 4 Gray Edges

We say that a reversal acts on a gray edge $e$ if it acts on the black edges incident with $e$. A gray edge $(\pi_i, \pi_j)$ is called oriented if it satisfies any of the following equivalent conditions,

1. The reversal acting on it results in $\Delta b - \Delta c = -1$.
2. The reversal acting on it conforms to cases 4a, 4b or 4d described above.
3. $i + j$ is even.
4. The signs of elements in the original permutation from where $\pi_i$ and $\pi_j$ came from, are opposite.
5. The egdes look as shown in figure 2 with respect to the black edges incident on it. This is clear from the above point.

The proof of this is obvious from figure 2.

A reversal is called oriented if it acts on an oriented gray edge. Other reversals are called unoriented.
5 The Overlap Graph $OV(\pi)$

Here we describe the construction of a graph that will be central to our discussion on finding a closed form expression for $d(\pi)$, the reversal distance of $\pi$. This graph is actually an interval overlap graph with vertices being the gray edges of $B(\pi)$. To each gray edge $(\pi_i, \pi_j)$ (vertex) associate the interval $[i, j]$. There will be an edge between two vertices if their corresponding intervals overlap but no one contains the other. In this case the gray edges are called intersecting. This graph will be refered to as $OV(\pi)$. An example is shown in figure 1c.

A vertex in this graph is called oriented if the corresponding gray edge is oriented. A connected component of $OV(\pi)$ is called oriented if any vertex in it is oriented.

6 Interleaving Graph $H_\pi$

Pevzner and Hannenhalli [4], who wrote the first paper giving the exact expression for $d(\pi)$ did not use the overlap graph but a similar graph. Therefore, a short description is called for. Two cycles $C_1$ and $C_2$ of $B(\pi)$ are called interleaving if there exits gray edges $g_1 \in C_1$ and $g_2 \in C_2$ such that $g_1$ and $g_2$ are intersecting. The interleaving graph (abbreviated $H_\pi = (C_\pi, I_\pi)$) has vertex set $C_\pi$ = set of all cycles of $B(\pi)$ and edge set $I_\pi = \{C_1, C_2\} | C_1$ and $C_2$ are interleaving cycles.

The concept of oriented edges and connected components are same as for above.

7 Relation Between $OV(\pi)$ and $H_\pi$

The result that we will need is that the connected components of $OV(\pi)$ and $H_\pi$ are the same with the same orientation. For any set $S$ of gray edges, define $min(S) = min\{i | (\pi_i, \pi_j) \in S\}$
and \(\max(S) = \max\{j \mid (\pi_i, \pi_j) \in S\}\) i.e. \(\min(S)\) is the smallest left endpoint of all edges in \(S\) and \(\max(S)\) is the largest right endpoint. Also define \(\text{span}(S) = [\min(S), \max(S)]\). Now we can prove the following lemmas,

**Lemma 7.1** : If \(M\) is the set of all gray edges of a connected component in \(B(\pi)\) then \(\min(M)\) is even and \(\max(M)\) is odd.

**Proof** : First note that there is no black edge between \(\pi_{2i-1}\) and \(\pi_{2i}\) for any \(i\). This is because of our transformation to convert signed permutation into an unsigned one and \(\pi_{2i-1}\) and \(\pi_{2i}\) differ by exactly one. Assume that \(\min(M)\) is odd. Therefore, one of \(\pi_{\min(M)} + 1\) and \(\pi_{\min(M)} - 1\) is to its immediate right, call this \(\pi_a\) and the other as \(\pi_b\). Since \(\pi_{\min(M)}\) has a gray edge incident on it, it must be to \(\pi_b\). Hence \((\pi_{\min(M)}, \pi_b) \in M\) and \(\pi_{\min(M)}\) is neither the maximum nor the minimum in the set \(\{\pi_i \mid i \in \text{span}(M)\}\). Let \(\pi_l\) be this minimum or maximum for some \(\min(M) < l < \max(M)\). This element does not have \(\pi_l - 1\), which lies outside \(\text{span}(M)\), adjacent to it and must have a gray edge going to it, but this contradicts the fact that no edge of \(M\) can leave \(\text{span}(M)\). Hence \(\min(M)\) is even. \(\max(M)\) can be proved to be odd in a similar fashion.

**Lemma 7.2** : Every connected component of \(OV(\pi)\) corresponds to the set of gray edges of a union of cycles.

**Proof** : This can be proved by proving that if \(M\) is a connected component and \(g\) is a gray edge of a cycle \(C\) then,

\[ g \in M \Rightarrow C \subseteq M \]

First note that spans of different components cannot overlap without one containing the other. This is because if an edge has one endpoint in \(\text{span}(M)\) and the other outside it then it must intersect with some edge of \(M\). So if \(C\) is a cycle which has its edges in two different components \(M_1\) and \(M_2\), pick two
consecutive gray edges $e_1 \in M_1$ and $e_2 \in M_2$ in $C$. Consider the following cases,

1. $\text{span}(M_1) \subseteq \text{span}(M_2)$. Since $e_1$ and $e_2$ are in different components, they cannot interleave. Now, $e_1$ and $e_2$ have a black edge between them but black edges only go from an even position to an odd position. Hence, either the left endpoint of $e_1$ is $\min(M_1)$ and odd or the right endpoint of $e_1$ is $\max(M_1)$ and even. In either case, we have a contradiction to Lemma 1.

2. $\text{span}(M_1)$ and $\text{span}(M_2)$ are disjoint: Assuming that $\max(M_1) < \min(M_2)$, the right endpoint of $e_2$ must be $\max(M_1)$ and even and the left endpoint of $e_2$ must be $\min(M_2)$ and odd. This also gives us a contradiction to Lemma 1.

This proves the claim we made at the beginning of this section. Note that the second lemma, in particular, implies the following.

**Theorem 7.1:** The overlap graph cannot contain isolated vertices.

### 8 Effect of a Reversal

Let us now characterize the effect of a reversal on a permutation. Note that the identity permutation $i$ is the only permutation with no edges in $B(\pi)$ and hence no vertices or edges in $OV(\pi)$. Therefore, our aim is to reduce the edges in $B(\pi)$ or the vertices in $OV(\pi)$. For reasons that will become clear later, we will be more interested in the intersection of gray edges and the graph $OV(\pi)$ provides us with that information. The paper [2] gives an important observation on how $OV(\pi)$ changes by application of reversals.
**Notation 8.1 :** Let $e$ be a gray edge. Let $r(e)$ denote the reversal acting on $e$, $N(e)$ denote the neighbouring vertices of $e$ in $OV(\pi)$ i.e. the gray edges intersecting with $e$.

**Theorem 8.1 :** Let $e$ be a vertex in $OV(\pi)$ and let $\pi' = \pi r(e)$. $OV(\pi)$ could be obtained from $OV(\pi)$ by the following operations.

1. Complement the graph induced by $OV(\pi)$ on $N(e) - \{e\}$, and flip the orientation of every vertex in $N(e) - \{e\}$.
2. If $e$ is oriented in $OV(\pi)$ then remove it from $OV(\pi)$.
3. If there exists an oriented edge $e'$ in $OV(\pi)$ with $r(e) = r(e')$ then remove $e'$ from $OV(\pi)$.

**Proof :** The proof for this theorem is very simple and we will show it graphically in figure 3. The first figure shows that if two intersecting edges are intersecting with $e$ then after the reversal $r(e)$, they stop intersecting. The opposite happens for edges intersecting with $e$ but not intersecting themselves. Suppose $f$ is a gray edge intersecting with $e$, then after the reversal $r(e)$, the sign (in the original permutation) of only one of the endpoint of $f$ changes. This flips the orientation of $f$. Hence, the first part of the theorem is proved.

The proof for the second and third part can be seen from figure 3.

**9 Hurdles and A Better Bound on $d(\pi)$**

We have seen that in the breakpoint graph $B(\pi)$ of a permutation $\pi$, there are two types of gray edges - oriented and unoriented. The graph $OV(\pi)$ is undirected and hence may be partitioned into connected components. Let us define a component $C$ in $OV(\pi)$ to be oriented if it contains atleast one oriented
vertex. Note that an oriented vertex in $OV(\pi)$ is one which corresponds to an oriented edge of $B(\pi)$. Any component which has only unoriented vertices (gray edges) is called unoriented.

**Notation 9.1:** We will denote unoriented edges (or components) by $U$ and oriented edges (or components) by $O$.

A natural question that might come to the mind is why we are classifying components. As it turns out, to find the optimum (or rather an optimum) sequence of reversals, we actually need to work with components, not individual edges.

### 9.1 The Inadequacy of $b(\pi) - c(\pi)$

We have already seen that $d(\pi) \geq b(\pi) - c(\pi)$. Further by referring to Theorem 3.1, we also find that in cases 4a, 4b and 4d, we can clearly attain the maximum possible decrease in $b(\pi) - c(\pi)$ i.e. 1. Then why not just continue applying reversals of this type to reduce $b - c$ by 1 at each step and hence attain the sorted permutation in just $b(\pi) - c(\pi)$ steps? The answer is obvious. Such reversals may not exist at all. That problem occurs if at any step in such a reduction, we are left with a permutation without an oriented gray edge. Thus this lower bound on the number of reversals needed is not good enough if we want to construct an algorithm for finding the optimum sequence of reversals. To do that, we introduce the essential concept of a hurdle.

### 9.2 Hurdles

The above discussion may have given an impression that oriented gray edges are good for us (we can simply take a reversal corresponding to such an edge to get $\Delta(b - c) = -1$ and thus
attain the steepest possible descent towards the identity permutation) and unoriented ones are not. That is not exactly true. Actually this is true of components. Oriented components are good, unoriented ones are not. In this section we try to characterise the unoriented components further.

If we consider any connected component of gray edges, the vertices of $B(\pi)$ in it can be partitioned into sets of consecutive elements of $\pi$. Note further that by the definition of a connected component, every vertex of $B(\pi)$ is contained in at most one connected component of gray edges (it may be in none if it has no gray edge incident on it, i.e. both its consecutive elements are adjacent to it). We have already defined the span of a connected component. It follows then that spans of two connected components are either disjoint or one contains the other. Thus we can define a partial order $\leq$ on connected components of gray edges in $\pi$ as follows.

**Relation $\prec$** : For two distinct connected components of gray edges, $C_1$ and $C_2$, $C_1 \prec C_2$ if $\text{span}(C_2)$ contains $\text{span}(C_1)$.

Now consider only the unoriented components of $\pi$. We first observe that the above partial order can be restricted to such components only. Actually, this relation is not needed on oriented components, so we will now use the notation $\prec$ to mean the above partial order restricted to unoriented components only. We next define another concept.

**Separation** : Let $U_1$ and $U_2$ be two unoriented components whose spans are disjoint. An unoriented component $U$ is said to separate $U_1$ and $U_2$ if it contains a vertex in $B(\pi)$ lying between the spans of $U_1$ and $U_2$. 

15
Now we can conveniently define a hurdle.

**Hurdle:** An unoriented component in $U$ is called a hurdle if it is

1. minimal with respect to $<$ i.e. its span contains no other unoriented component or
2. maximum with respect to $<$ but does not separate any two unoriented components i.e. its span contains all hurdles but the vertices contained in it are either to the left or right of all other unoriented components.

The above characterisation of a hurdle may not actually explain what it is, so we give an alternate characterisation.

**Theorem 9.1:** Consider placing on a circle all vertices of $B(\pi)$ that are contained in some unoriented component of gray edges. Any unoriented component $U$ is a hurdle in $\pi$ iff all vertices contained in it lie consecutively on the circle.

**Proof:** First suppose that $U$ is a hurdle. So it is either minimal or maximum in $<$. If it is minimal and its span (on the circle) also contains some vertex not in it, then let this vertex be in component $U_1$. Clearly then either $U_1 < U$ or $U$ and $U_1$ intersect. The former violates the minimality $U$ and the latter contradicts the definition of a connected component. If $U$ is maximum, then its span on the circle will contain the elements 0 and $2n + 1$ (this may not be true if 0 and $2n + 1$ are not in unoriented components but what we mean here is that the span of $U$ on the circle starts towards the end of $B(\pi)$ and then wraps around $2n + 1$ to 0). If this span contains a vertex not in $U$, then as before, if $U_1$ contains the vertex, either $U$ or $U_1$ intersect or $U < U_1$. Either way we have a contradiction. Conversely, it is easy to see that if the span of $U$ is continuous on the circle then $U$ is minimal if the span does not have 0 and
2n + 1 (the span does not wrap around) and maximum otherwise.

Please see figure 4 for a description of hurdles.

9.3 A Better Bound on $d(\pi)$

Having talked of connected components of $OV(\pi)$ and hurdles, we can now give a better lower bound on the value of $d(\pi)$. First we need some simple definitions and lemmas.

**Notation 9.1**: $h(\pi)$ denotes the number of hurdles in permutation $\pi$.

**Lemma 9.1**: Let $e = (\pi_k, \pi_l)$ be a gray edge. Then a reversal $\rho(2i - 1, 2j)$ not corresponding to $e$ reverses the orientation of $e$ iff exactly one of $k$ and $l$ is contained in $[2i - 1, 2j]$.

**Proof**: First observe that for any $t \in [2i - 1, 2j]$, $\rho$ flips the parity of $\pi_t$ i.e. after the reversal, $\pi_t$ is mapped to $\pi_s$ such that $s + t$ is odd. If this condition is untrue, $\pi_t$ is untouched. Now suppose $e$ is unoriented and only $k \in [2i - 1, 2j]$. Then $k + l$ is odd. Thus by what we just discussed $e$ gets mapped to $(\pi_s, \pi_l)$ where $s + k$ is odd i.e. $s + l$ is even and hence $e$ becomes oriented. The remaining cases of the proof are similar.

**Lemma 9.2**: Let $U$ be an unoriented component. Then $\rho(2i - 1, 2j)$ destroys $U$ i.e. causes it to stop existing or changes its orientation iff for some edge $u = (\pi_k, \pi_l) \in U$, exactly one of $k$ and $l$ is in $[2i - 1, 2j]$ or $\rho$ corresponds to some gray edge of $U$.

**Proof**: If such a $u$ exists, then by the above Lemma, it will become oriented. Thus if $U$ does not break as a result of this reversal, it will become oriented. In either case, it is destroyed.
If \( \rho \) corresponds to an edge of \( U \), the edge will also intersect another edge of the same unoriented component, by Theorem 7.1. This edge will now become oriented and hence the component becomes oriented. If conversely, no such \( u \) exists and \( \rho \) does not correspond to an edge of \( U \), each edge of \( U \) remains unoriented after the reversal (by the previous Lemma) and hence \( U \) remains unoriented.

**Lemma 9.3**: Any reversal \( \rho(2i - 1, 2j) \) can destroy at most two hurdles of \( \pi \), so that \( \Delta h \geq -2 \).

**Proof**: First observe that from the previous Lemma it follows that \( \rho \) can destroy a minimal hurdle iff at least one of \( 2i - 1 \) and \( 2j \) is in the span of the hurdle. Thus a reversal can destroy at most two minimal hurdles. If it destroys only one minimal hurdle, we have \( \Delta h \geq -2 \), irrespective of whether it destroys the only possible maximum hurdle or not. If it destroys two minimal hurdles, one must contain the point \( 2i - 1 \) and the other \( 2j \). Thus to destroy the maximum hurdle as well, the maximum hurdle would have to have a vertex \( v \) of \( B(\pi) \) between \( 2i - 1 \) and \( 2j \), excluding both (Lemma 9.2). Then clearly if \( H_1 \) and \( H_2 \) are the destroyed minimal hurdles, then \( v \) lies between their spans and hence the maximum hurdle separates them. This is a contradiction to the definition of a maximum hurdle. Hence \( \rho \) cannot destroy the maximum hurdle. So in all cases, \( \Delta h \geq -2 \).

**Lemma 9.4**: For any reversal \( \rho(2i - 1, 2j) \), \( \Delta(b - c + h) \geq -1 \).

**Proof**: From Theorem 3.1 it follows that \( t = \Delta(b - c) \in \{1, 0, -1\} \). If \( t = -1 \), then by the same theorem, \( \rho \) acts on an oriented cycle and hence this cycle is not part of any unoriented component (Lemma 7.2). Further, if any edge \( u \) crosses this cycle, then the component \( U \) containing \( u \) is oriented by definition. By Lemma 9.2 it follows that this reversal cannot affect or destroy any unoriented component, let alone a hurdle. Thus
$$\Delta h = 0.$$ Therefore, $$\Delta(b - c + h) = -1.$$ If $$t = 0$$ then by Theorem 7.1, it follows that we are dealing with case 4c (the only case with $$\Delta(b - c) = 0$$). In that case, the reversal acts on a gray edge, hence the vertices $$\pi_{2i-1}$$ and $$\pi_{2j}$$ are part of the same connected component, so that the proof of Lemma 9.3 shows that this reversal destroys atmost one hurdle. Thus $$\Delta h \geq -1$$ and hence $$\Delta(b - c + h) \geq -1.$$ Finally if $$t = 1$$, then the result trivially follows from Lemma 9.3.

Having proved these Lemmas, we come to the main result which was originally proven by Hannenhalli and Pevzner.

**Theorem 9.2[HP] :** For any arbitrary signed permutation $$\pi$$, 
$$d(\pi) \geq b(\pi) - c(\pi) + h(\pi).$$

**Proof:** This follows immediately by observing that for the identity permutation, $$b - c + h = 0$$ and by Lemma 9.4, no reversal can reduce this value by more than 1.

## 10 Safe and Proper Oriented Reversals

Having studied the importance of the value $$\Delta(b - c + h)$$, we define safe and proper reversals.

**Proper Reversal:** A reversal $$\rho$$ is called proper if $$\Delta(b - c) = -1$$.

Note that by Theorem 3.1, it follows that the reversal corresponding to every oriented gray edge is proper. Thus any permutation with an oriented component has a proper reversal. Now we define a stronger concept.

**Safe Reversal:** A reversal $$\rho$$ is called safe if $$\Delta(b - c + h) = -1$$. 
Note that since $\Delta(b - c + h) \geq -1$, it is in our best interest to choose safe reversals at each step, so as to reach the identity permutation in minimum number of steps. It should be clear that if we can prove that every permutation has a safe reversal then $d(\pi) = b(\pi) - c(\pi) + h(\pi)$. That is unfortunately not true. We can however show that $d(\pi) \leq b(\pi) - c(\pi) + h(\pi) + 1$. That is what we will try to establish next. First we establish a relation between the existence of oriented components and safe reversals which will also explain our emphasis on oriented components.

### 10.1 Existence of Safe and Proper Reversals in Oriented Components

Now we try to establish that every permutation with atleast one oriented edge (equivalently, atleast one oriented component) has a reversal which is safe and proper. Note that for a safe and proper reversal, $\Delta(b - c) = -1$ and $\Delta h = 0$. Thus such a reversal must act on an oriented cycle and leave the hurdles untouched. That explains why such a reversal cannot exist in a permutation that has only unoriented components. To actually establish the existence of such reversals in permutations with oriented edges, we present two proofs. The first one is due to Kaplan, Shamir and Tarjan and the second due to Anne Bergeron. Later we will convert both these proofs into algorithms for computing optimum sequences of reversals.

**Happy Clique** : Consider a clique $C$ of oriented vertices in $OV(\pi)$. A clique here corresponds to a set gray edges, each of which intersects every other. $C$ is called happy if for every oriented vertex (gray edge) $e \notin C$ and every $f \in C$ such that $(e, f) \in OV(\pi)$ ($e$ and $f$ intersect), we have an oriented vertex $g \notin C$ such that $(g, e) \in OV(\pi)$ and $(g, f) \notin OV(\pi)$.
By Theorem 8.1, it follows that if we perform the reversal corresponding to \( f \), then even though \( e \) is unoriented, the component containing it also has \( g \) and \( g \) remains oriented, so that this component also remains oriented. Thus this reversal cannot create an unoriented component that did not already exist. Hence it cannot create a hurdle. All that we now need to show is that such a happy clique exists. Before that we formalise the argument that a happy clique \( C \) does indeed have a vertex (grey edge) \( e \) in it such that the reversal corresponding to \( e \) is safe and proper (it is proper by definition as \( e \) is oriented).

**ON(\( e \)), UN(\( e \))** : For an edge \( e \), let \( UN(e) \) be the set of unoriented vertices adjacent to \( e \) in \( OV(\pi) \) and \( ON(e) \) be the set of oriented neighbours. Finally, let \( N(e) \) be the set of all neighbours.

**Theorem 10.1[KST]** : Let \( C \) be a happy clique in \( OV(\pi) \) and let \( e \) be a vertex in \( C \) such that \( |UN(e)| \geq |UN(e')| \) for every vertex \( e' \) in \( C \). Then the reversal \( r(e) \) corresponding to \( e \) is safe.

**Proof** : First we observe that by Theorem 3.1, the reversal corresponding to \( e \) affects only the component containing \( e \), say \( M \). Then suppose that after application of \( r(e) \), this component splits into many others, say \( M_1(e), M_2(e), \ldots, M_k(e) \). Suppose that \( M_i(e) \) is unoriented. Clearly, by the same theorem, \( S = N(e) \cap M_i(e) \neq \emptyset \). Suppose that for some \( y \in S \), \( y \notin C \). Then since \( y \) is now unoriented, it must have been oriented earlier, so that by the definition of a happy clique, there is also some other oriented vertex \( g \) such that \( g \) is adjacent to \( y \) but not to \( e \). Thus \( g \) remains oriented and hence \( M_i(e) \) is oriented which is a contradiction. Thus we must have \( y \in C \) which implies that \( S \subseteq C \).

Now suppose \( y \in S \) and \( z \in UN(e) \). Then clearly, after the
reversal, $z$ is oriented and hence at that point, $y$ and $z$ are not adjacent. So they must have been adjacent in $OV(\pi)$. Thus $UN(e) \subseteq UN(y)$. Now by Theorem 7.1, $y$ is not the only vertex of $M_i(e)$. So let $x$ be a neighbour of $y$ in $M_i(e)$. Since $N(e) \cap M_i(e) \subseteq C$, $x$ is not a neighbour of $e$ in $OV(\pi)$. Thus it must be unoriented in $\pi$ as well. So then we must have $x \in UN(y)$ but $x \not\in UN(e)$. This implies $UN(e) \subset UN(y)$ and hence we contradict the assumption that $|UN(e)| \geq |UN(y)|$. So we must have $M_i(e)$ oriented for all $i$. In particular, the reversal $r(e)$ does not create new unoriented components or hurdles. So $\Delta h = 0$. Further, $\Delta(b-c) = -1$ as $e$ is oriented. So the reversal is safe and proper.

Now we come to proving that such a happy clique indeed exists if the permutation $\pi$ has an oriented edge.

**Theorem 10.2 [KST]**: Let $e$ be an oriented vertex in $OV(\pi)$. Then there exists a happy clique $C \subseteq ON(e)$.

**Proof**: Let $Ext(e)$ be the set of all neighbours of $e$ which have oriented neighbours outside $ON(e)$. We have two cases now:

1. $Ext(e) = ON(e) - \{e\}$. Here we can take $C = \{e\}$. Clearly, then $C$ is a happy clique.

2. $Ext(e) \subset ON(e) - \{e\}$. Set $D^0 = ON(e) - Ext(e)$. Now while $D^i$ is not a clique, let $K^i$ be a maximal clique of $D^i$ and define $D^{i+1} = D^i - K^i$. Then the final clique so obtained is a happy clique. Note that since we are removing a maximal clique at every step, if ever we get $M^i = \phi$, then $M^{i-1}$ is a clique, so that the step is not valid. Hence what we have a non-empty clique. To prove that it is happy, we can show by induction that each set $M^i$ satisfies the happy clique property.

**Theorem 10.3** : Every permutation with an oriented gray
edge (oriented component) has a safe and proper reversal that
does not affect any other component in $OV(\pi)$ and does not
create new unoriented components.

**Proof**: The proof follows immediately from Theorems 10.2,
10.1 and 3.1 and proof of Theorem 10.1.

The above theorems are one way of proving this result. There
are others as well. Given below is another proof (due to Anne
Bergeron).

**Score**: Define the score of a reversal as the number of unori-
ented edges in the permutation resulting from the reversal.

From Theorem 3.1 it follows that for an oriented edge $e$, if
$T$ is the number of existing unoriented vertices, $score(r(e)) =
T - |UN(e)| + |ON(e)|$.

**Theorem 10.4 [Adapted from ANB]**: Of all oriented edges,
let $e$ have minimal score. Then the reversal corresponding to $e$
is safe (and proper).

**Proof**: Suppose that the reversal corresponding to $e$ creates
an unoriented component $C$. Then at least one vertex of $C$ was
adjacent to $e$ before the reversal. Suppose we call this $w$. Note
that $w$ is now unoriented, hence earlier it was oriented. First we
observe that if $y \in UN(e)$, then after the reversal it is oriented
and hence to ensure that $C$ is unoriented, it is not adjacent to
$w$ after the reversal. Thus before the reversal, it is adjacent
to $w$ and we have $y \in UN(w)$. Thus $UN(e) \subseteq UN(w)$. In a
similar manner, $ON(w) \subseteq ON(e)$. Further, if $UN(e) = UN(w)$
and $ON(e) = ON(w)$, then clearly $e$ and $w$ have the same set
of neighbours, so that $w$ is isolated after the reversal. Since isolated
vertices do not exist in $OV(\pi)$, $w$ must disappear, a contradic-
tion. This means that $|ON(e)| - |UN(e)| > |ON(w)| - |UN(w)|$,
which contradicts the minimality of $score(e)$. Hence $C$ must be oriented and as in the proof of theorem 10.1, the reversal corresponding to $e$ is both safe and proper.

The above Theorem thus is another proof of the fact that in every permutation with an oriented edge, there exists a reversal which is safe and proper. We formalise this result below.

**Theorem 10.5**: Given a permutation $\pi$ with an oriented edge $e$, there exists a sequence of safe and proper reversals, none of which creates an unoriented component such that the resulting permutation, $\pi'$ has no more oriented edges. Further, the unoriented components of $\pi'$ are same as those of $\pi$.

**Proof**: To prove this, we observe that we can repeatedly apply Theorem 10.3, till we are left without any oriented gray edges. Note that we can also use the method of Theorem 10.4 repeatedly to the same effect.

The above theorem almost completely explains our line of attack. We begin with a permutation and keep applying safe and proper reversals on its oriented edges, till all oriented edges are gone. Note that till now we have done the best possible by reducing $(b - c + h)$ by 1 at each step. When we are left with only unoriented components, we cannot use these results. Then we use some other method to find a safe (but not proper) reversal. Having applied this, we might get oriented components again. If so, we can again use these results till we again exhaust oriented edges and then re-use our second method. This continues till we are done.

Alternatively, if we can remove all unoriented components of $\pi$ in the beginning, then we are left with oriented components only. Then Theorem 10.5 guarantees that we can get to the identity
permutation using only safe and proper reversals. The first of these approaches was followed by Hannenhalli and Pevzner, who gave the first polynomial time algorithm for this problem. Subsequent researchers have used the second approach as it clearly divides the problem into two parts - removing unoriented components and then using methods like the above to remove oriented edges. We turn next to removing unoriented components, which brings us back to hurdles.

11 Dealing with Unoriented Components

When we were talking of reversals on oriented components, we were always using proper reversals i.e. those that act on oriented edges and satisfy $\Delta(b - c) = -1$. When we talk of reversals on unoriented components, we are talking about safe but improper reversals. For such reversals, $\Delta(b - c) \geq 0$, so that $\Delta h \leq -1$. Thus such reversals must always destroy atleast one hurdle. We consider these cases separately. The first one is where we destroy exactly one hurdle (without adding a new one). The second one is where we destroy two hurdles at a time.

11.1 Hurdle Cutting

The technical name for destroying exactly one hurdle through a safe reversal is hurdle cutting. Note that since $\Delta h = -1$, this forces $\Delta(b - c) = 0$ for a safe reversal. Thus we are dealing with case 4c of Theorem 3.1. First note that by definition, a hurdle is a minimal (or a maximum) unoriented component and cutting it only requires us to choose a gray edge $e$ contained in it and using the reversal corresponding to that. Note that since the gray edge of no other component can intersect this edge $e$, by Theorem 7.1, it follows that an edge $f$ of the hurdle itself will intersect $e$. Then by Theorem 8.1 it follows that $f$ will become
oriented after the reversal, so that the hurdle will cease to exist. Further note that if this hurdle \( H \) splits into new components \( U_1, \ldots, U_k \), then each of these will have at least one oriented vertex, so that no unoriented component is created. Finally observe that Theorem 8.1 guarantees that no component other than the hurdle is affected by the reversal. Hence we have successfully created a reversal with \( \Delta (b - c) \leq 0 \), destroying a hurdle and creating no new unoriented component. But does that guarantee that \( \Delta h = -1 \) and hence \( \Delta (b - c + h) \leq -1 \)? Unfortunately, not. Why? Well what if this hurdle were the only unoriented component contained in another unoriented component \( U \). Note that \( U \) was not a hurdle as it contained \( H \). Now that \( H \) has split into oriented components, \( U \) has become a hurdle. Thus \( \Delta h = 0 \) in this very bad case. To formalise this notion, we define a superhurdle.

**Superhurdle:** A hurdle is called a superhurdle if cutting it without affecting any other component creates another hurdle.

Note that a hurdle is a superhurdle iff it is minimal and is the only unoriented component contained in another unoriented component \( U \) or it is maximum and contains another unoriented component which spans all other unoriented components, without separating them. In the circle notation introduced in Theorem 9.1, this means that a hurdle is a superhurdle iff its span (which is continuous on the circle) separates the only two continuous portions of the span of another unoriented component.

In the first paragraph of this subsection, we essentially proved the following theorem.

**Theorem 11.1:** If a hurdle \( H \) is not a superhurdle, then the
reversal corresponding to any gray edge in it is safe. Further, such a reversal will cut the hurdle, not create any new unoriented components and not affect any other component.

11.2 Hurdle Merging

Destroying two hurdles through a reversal is called hurdle merging. Note that hurdle merging is a very different operation from what we have considered so far. When we talked of oriented safe reversals or hurdle cutting, we were dealing with exactly one connected component. We just picked some suitable gray edge of the component and then performed the corresponding reversal. That ensured that no other components were affected. When we talk of hurdle merging, we are talking of two separate components and hence the reversal must contain end points in two different components. This means that we would affect many components of the overlap graph, $OV(\pi)$.

When we talk of hurdle merging, we will always talk of case (3) of Theorem 3.1, which ensures (as in the proof of this theorem), that both these hurdles merge into one. That will happen because the black edges we choose to cut for our reversal will merge into one cycle. Further, as the proof of the theorem shows, the resulting cycle is oriented.

This has some side effect as well. First, if these two were the only unoriented components contained in another unoriented component, then that will become a hurdle now. To avoid that we will always ensure that when we merge two hurdles, we merge non-consecutive ones. In the circle description, this means that we will look at hurdles which have another hurdle contained between them, no matter what direction we traverse the circle in. Then this reversal will not affect that hurdle and hence we
will not create a new hurdle. Second, and more important, if some oriented edge \( e \) has exactly one end point in the span of reversal, it will become unoriented now. So that can create new unoriented components. But it does not! Why? That is because we just said that the two hurdles will merge to form an oriented component. Clearly, then there must be sequence of intersecting edges of this new component that spans the whole span of the reversal. Thus this now unoriented edge \( e \) must intersect one of these and hence, will become part of this new merged oriented component. Consequently, we do not create new unoriented components. Note however, that we do affect other components. But the affect is limited only to changing them from unoriented to oriented or breaking them into two (or more) oriented components.

To summarise we just observe that if we pick up hurdles which are non-adjacent on our circle, choose one black edge in each one of them and perform the reversal that breaks both of these, then at the end, we would have merged both these into one oriented component, possibly created new oriented components (but no unoriented components) and affected no other hurdle. Further, we would have \( \Delta(b - c) = 1 \) from Theorem 3.1 and \( \Delta h = -2 \) to give \( \Delta(b - c + h) = -1 \), so that this reversal is safe. We also observe here that we can always find two such non-adjacent hurdles iff \( h > 3 \). Thus we have in essence proved the following theorem.

**Theorem 11.2 :** If for a permutation \( \pi \), \( h > 3 \), then there is a safe reversal that destroys two of these hurdles, does not create new unoriented components and affects no other hurdle. Further, no such safe reversal would affect any other hurdle or change a superhurdle to a non-superhurdle.

**Proof :** We proved the first part of this theorem above. To prove the second part, we only need to observe that spans of
hurdles do not intersect on the circle. If further, $H$ is a super-hurdle, then there is another unoriented component $U$ whose span contains only $H$ and hence the span of $U$ cannot intersect the hurdles we are merging. Thus the merge affects neither $U$, nor $H$ implying that $H$ remains a super-hurdle.

We have seen so far what to do if we can find a non-super-hurdle to cut or two non-adjacent hurdles to merge. Note that the latter case applies iff $h(\pi) > 3$. As it turns out, if $h = 1$, then there is exactly one unoriented component, so that this single hurdle cannot be a super-hurdle and we can safely cut it.

**Theorem 11.3 :** If a permutation has exactly one hurdle, then that is the only unoriented component of the permutation. In that case, we can safely cut it.

**Proof :** To prove this, we only observe that if we take any un-oriented component, then it is either a minimal hurdle or must contain a minimal hurdle. So that if we have atleast two non-intersecting unoriented components, then we must have atleast two distinct minimal hurdles, a contradiction. Thus all hurdles are contained in one another. Then clearly we must have a maximum hurdle. Thus the only way out is that the only maximum and minimal hurdle coincide i.e. there is only one unoriented component. Thus this is not a super-hurdle and by Theorem 11.1, we can safely cut it.

Now if $h(\pi) = 2$, then we can prove on similar lines that merging them is safe.

**Theorem 11.4 :** If a permutation has exactly two hurdles, then merging them is safe.

**Proof :** Consider our circle notation. First observe that if there is any unoriented component, not containing one of these
two hurdles, then it is either a hurdle or contains another distinct hurdle, which is not possible. So every unoriented component must contain one of these hurdles. That means that for any unoriented component \( U \), different from the two hurdles, there exists an edge \( e \in U \) which has one endpoint between the hurdles and one outside them. Then, clearly merging the hurdles will cause this edge to become oriented and hence \( U \) is also destroyed (split into oriented components) by the reversal.

At the end the only case that remains is \( h(\pi) = 3 \). Unfortunately, here the safe reversal need not exist and it is this case that gives us the value \( d(\pi) = b(\pi) - c(\pi) + h(\pi) + 1 \), instead of one less. We will return to this later.

12 Closed Form Expression for \( d(\pi) \)

In this section, we find a closed form expression for the value of \( d(\pi) \). The proofs presented here will be used for developing algorithms later.

12.1 When Things are Good!

Let us now try to find a closed form expression for the value of \( d(\pi) \) in good cases. All cases, except for one are good.

**Theorem 12.1:** If \( \pi \) is a permutation with an even value of \( h(\pi) \), then \( d(\pi) = b(\pi) - c(\pi) + h(\pi) \).

**Proof:** We give a constructive proof. First we repeatedly apply Theorem 11.2 to find safe hurdle merging reversals and reduce the number of hurdles by two at each step. Note that this Theorem guarantees that new unoriented components are not added. Finally we are left with two hurdles. Then using Theorem 11.4, we can perform another safe merge to get a
permutation without any hurdles and hence without unoriented components. At this point, let the permutation be \( \pi' \). Then \( \pi' \) satisfies \( h = 0 \). Now we use Theorem 10.5 to get a permutation without any oriented components and without adding new unoriented components. Clearly since \( \pi' \) had no unoriented components, this final permutation has no components at all. Thus this is the identity. So at the end \( (b - c + h) = 0 \). Further we have reduced this value by 1 at each step, which implies that we used exactly \( b(\pi) - c(\pi) + h(\pi) \) steps. Together with Theorem 9.2 this guarantees that this sequence was indeed optimum.

**Theorem 12.2 :** If \( \pi \) is a permutation with odd \( h(\pi) \) and atleast one hurdle of \( \pi \) is not a superhurdle then, \( d(\pi) = b(\pi) - c(\pi) + h(\pi) \).

**Proof :** We observe that using Theorem 11.1, we can first perform a safe hurdle cutting to get another permutation with an even number of hurdles. Then our previous theorem immediately gives us the result.

The above two theorems imply that unless \( \pi \) has an odd number of hurdles, each of which is a superhurdle, we have \( d(\pi) = b(\pi) - c(\pi) + h(\pi) \). Permutations with an odd number of hurdles, each of which is a superhurdle are called fortresses (they are “hard” to “break”). These require one more reversal, which is the subject of the next section.

### 12.2 When the Going Gets Tough - Fortresses

Let us begin by defining a fortress.

**Fortress :** A fortress is a permutation with an odd number of hurdles, each of which is a superhurdle. (This definition implies that \( h \geq 3 \))
**3-Fortress**: A 3-fortress is a permutation with exactly 3 hurdles, each of which is a superhurdle.

We begin by proving a simple Lemma, which will be needed later.

**Lemma 12.1**: Let $\pi$ be a permutation which cannot be sorted by safe reversals alone. Then $d(\pi) \geq b(\pi) - c(\pi) + h(\pi) + 1$.

**Proof**: This follows from the fact that $d(\pi) \geq b(\pi) - c(\pi) + h(\pi)$ and this bound is achieved iff at each step we apply only safe reversals.

**Lemma 12.2**: Let $\pi$ be a 3-fortress. Then any reversal on $\pi$ destroying any existing hurdle is unsafe.

**Proof**: First we observe that in our circle notation, if $\pi$ is a fortress, then every hurdle is contained in an unoriented component containing only that hurdle. Now if any reversal destroys two existing hurdles, then its two endpoints lie in each of the two hurdles and hence, this reversal will destroy any unoriented component which contains any of these two hurdles. Consequently at least one unoriented component containing the third hurdle will now become a hurdle (such a component must exist as the third hurdle is also a superhurdle). This shows that $\Delta h \geq -1$.

Now consider a reversal destroying a hurdle. Suppose both endpoints of the reversal are in the span of this hurdle. In that case we must be in case 1, 2, 3 or 4 of Theorem 3.1. Then $\Delta (b - c) \geq 0$. Further, since this hurdle is a superhurdle, $\Delta h = 0$. Hence, $\Delta (b - c + h) \geq 0$, and the reversal is unsafe. If one endpoint of the reversal is in a hurdle, and the other in a different component, then we are in case 1, 2 or 3 of Theorem 3.1 and hence $\Delta (b - c) = 1$. But, $\Delta h \geq -1$, implying that the reversal is unsafe.
Theorem 12.3: Any sequence of reversals reducing a 3-fortress to the identity permutation has an unsafe reversal.

Proof: To prove this we observe that to get to the identity permutation, at some stage we would have to destroy the hurdles. Then the first reversal which destroys a hurdle is unsafe by the above lemma.

Theorem 12.4: Any sequence of reversals converting a fortress to the identity permutation contains an unsafe reversal.

Proof: We do this by induction on the number of hurdles in the fortress. For the base case $h = 3$ and the above theorem proves our result. So let this result be true when we have less than $k$ hurdles in a fortress. Now consider a fortress with $k$ hurdles, each of which is a superhurdle. If the first reversal has endpoints in two different connected components, then by Theorem 3.1, $\Delta(b - c) = 1$. So for the reversal to be safe, we need $\Delta h = -2$ i.e. the two end points of the reversal lie in two hurdles. Then such a reversal will destroy both these hurdles without affecting any other, so that we get a fortress of two less hurdles and we are done by induction. If the end points of the reversal lie in one connected component, which is oriented, the reversal affects only that component and then this argument repeats till we either reduce the hurdles or are left with no more oriented components. If the endpoints of the reversal lie in the same unoriented component, we are talking of cases 1, 2, 3 or 4(c) of Theorem 3.1 and hence $\Delta(b - c) \geq 0$. Further note that since we affect only one component, we are destroying atmost one hurdle. If this reversal destroys a hurdle, it will also create a new one (superhurdle property). If it does not act on a hurdle, $\Delta h = 0$. In any case $\Delta h = 0$. Thus this reversal is unsafe.
Having established these results, we are now in a position to provide a closed form expression for the number of reversals needed for a fortress.

**Lemma 12.3** : If $\pi$ is a fortress, $d(\pi) \geq b(\pi) - c(\pi) + h(\pi) + 1$.

**Proof** : Follows immediately from Theorem 12.4 and Lemma 12.1.

**Theorem 12.5** : If $\pi$ is a fortress, $d(\pi) = b(\pi) - c(\pi) + h(\pi) + 1$.

**Proof** : Given, a fortress, we apply the following sequence of steps to it. First, we keep merging hurdles according to Theorem 11.2. That guarantees that we are applying safe reversals i.e. reducing $b - c + h$ by 1 at each step. Finally, we would be left with exactly 3 hurdles, each of which is a superhurdle. Now we pick up black edges in any two of these hurdles and apply a reversal that breaks both of them. This corresponds to case 3 of Theorem 3.1 and hence, $\Delta(b - c) = 1$. Further, we have destroyed both hurdles, creating only oriented components. But at the same time, proof of Lemma 12.2 shows that we would also have converted exactly one unoriented component containing the third hurdle to a hurdle. Thus $\Delta h = -1$. After this, we are left with two hurdles, which we merge in accordance with Theorem 11.4. Finally we are left with only oriented components and Theorem 10.5 guarantees that a sequence of safe reversals leading to the identity permutation does indeed exist. In the whole process, we have reduced $b - c + h$ by 1 everywhere except at one point where this value was maintained. Thus this sequence of reversals has exactly $b(\pi) - c(\pi) + h(\pi) + 1$ steps. This shows that $d(\pi) \leq b(\pi) - c(\pi) + h(\pi) + 1$. Together with the above lemma, this completes our proof.
12.3 Putting it Together

Having done all this we can now compile all results together to give the main theorem which summarises all these results. The original proof of this theorem was given by Hannenhalli and Pevzner [4].

**Theorem 12.6[HP]** : Let $\pi$ be a signed permutation. Then

$$d(\pi) = \begin{cases} 
  b(\pi) - c(\pi) + h(\pi) & \pi \text{ is not a fortress} \\
  b(\pi) - c(\pi) + h(\pi) + 1 & \pi \text{ is a fortress}
\end{cases}$$

13 Algorithms

With all these results behind us, we are now ready to complete the objective of this report. This section describes the algorithms to solve the sorting by reversals problem. The algorithms and their complexities are as follows,

**KST** This algorithm is based on finding *safe reversals* through happy cliques. It takes $O(n\alpha(n))$ time to remove the hurdles and after that it takes $O(n)$ time to find a safe reversal resulting in a running time of $O(r'n + n\alpha(n))$ where $r'$ is the number of reversals required to sort the permutation after removing its hurdles.

**ANB** This algorithm is based on finding *safe reversals* by finding the reversal that maximises the number of oriented components. This removes the hurdles using the same procedure as for above. After the hurdles are removed, each reversal is found in $O(n^2)$ time giving a total running time of $O(r'n^2 + n\alpha(n))$.

**OUR** We have developed an algorithm based on the above idea but improve the running time from $O(n^2)$ per reversal to
\(O(n \log(n))\) per reversal. Hence our algorithm runs in \(O(r'n \log(n) + n\alpha(n))\) time.

To get an idea of how large \(r'\) can be, we can see that any signed permutation \(\pi = (\pi_1, \ldots, \pi_n)\) can be sorted using \(2n\) reversals. This can be done by using the first reversal to get 1 into position, then the second to change the sign of 1 (if needed) to +. Then the third to get 2 into position and so on. Hence \(d(\pi') \leq 2n\) where \(\pi'\) is the transformed version of \(\pi\) (Note that \(\pi'\) will have \(2n + 2\) elements). In fact, it has been shown that the maximum number of reversals required to sort a signed permutation on \(n > 3\) elements is \(n+1\) and this bound is tight. Hence \(r' = O(n)\). We begin by describing the algorithm given by [KST].

14 Algorithm given by [KST]

14.1 Overview of the Algorithm

Since the graph \(OV(\pi)\) has \(O(n^2)\) edges, trying to maintain it would be expensive. Therefore, the graph is represented implicitly by the permutations \(\pi\) and \(\pi^{-1}\). Using these, we can find out things like whether \(\pi_i\) has a gray edge incident to it or not, finding an endpoint of a gray edge given the other etc. in constant time. The advantage that this representation offers is that a reversal can be actually carried out in \(O(n)\) time by remaking \(\pi\) and \(\pi^{-1}\) where doing a reversal on \(OV(\pi)\) would have taken \(O(n^2)\) time in the worst case. We start the description of the algorithm by giving the scheme to find a happy clique.

14.2 Finding a Happy Clique

Let \(e_1, \ldots, e_k\) be the oriented edges of \(B(\pi)\) in increasing left endpoint order. The algorithm will traverse these edges in order and at the \(i^{th}\) step it will have a set \(C_i\) of edges which form a
happy clique in \(e_1, \ldots, e_i\). It will also keep the minimum span gray edge that contains all the gray edges in \(C_i\), if it exists. Let \(L(e)\) denote the left endpoint of an edge \(e\) and \(R(e)\) denote its right endpoint. Also let the set \(C_i = \{e_{i_1}, e_{i_2}, \ldots, e_{i_j}\}\). The invariant that this algorithm follows is,

**Invariant**: For every (oriented) edge \(e_l \notin C_i, l \leq i\),

1. If \(L(e_l) > L(e_{i_i})\), then \(t_i\) must be defined and \(e_l\) must be intersecting with it.

2. If \(L(e_l) < L(e_{i_i})\) and \(e_l\) is intersecting with a vertex in \(C_i\) then it is either intersecting with an edge \(e_p\) such that \(R(e_p) < L(e_{i_i})\) or it intersects with \(t_i\).

**Theorem 14.1**: The algorithm following the above invariant will produce a happy clique.

**Proof**: Let \(C\) be the clique and \(t\) be the minimal edge containing \(C\) produced after the procedure finishes. For any \(e_l \notin C\), it will either intersect \(t\) or another edge \(e_p\). By their definitions, both \(t\) and \(e_p\) are not in \(C\) and do not intersect with any edge in it as well. Hence \(C\) is a happy clique.

A trivial but important observation that will come in handy (and also shows the importance of keeping \(t_i\)) is as follows,

**Observation**: If \(e_l \notin C_i\) is a gray edge intersecting \(t_i\) then it satisfies the invariant irrespective of its left endpoint.

**Algorithm**: Start with \(C_1 = \{e_1\}\) and \(t_1\) as not defined. If after \(i\) steps, we are looking at the edge \(e_{i+1}\) then consider the following cases,

1. If \(R(e_{i_j}) < L(e_{i+1})\) then stop the algorithm and output \(C_i\) as the happy clique. This is correct because no other edge
is adjacent to $C_i$ (see figure 5a). Therefore, in the rest of the cases assume $L(e_{i+1} \leq R(e_i))$.

2. If $t_i$ is defined and $R(t_i) < R(e_{i+1})$ then $C_{i+1} = C_i$ and $t_{i+1} = t_i$ (See figure 5b). The edge $e_{i+1}$ satisfies the invariant because it is intersecting with $t_i$.

3. If $t_i$ is not defined or $R(e_{i+1}) \leq R(t_i)$.

(a) $R(e_{ij}) < R(e_{i+1})$ and $L(e_{i+1}) \leq R(e_{ii})$. Set $C_{i+1} = C_i \cup \{e_{i+1}\}$ and $t_{i+1} = t_i$ (See figure 5c). All edges that were satisfying the invariant before will also satisfy it now and $C_{i+1}$ is still a clique.

(b) $R(e_{ij}) < R(e_{i+1})$ and $L(e_{i+1}) > R(e_{ii})$. Set $C_{i+1} = \{e_{i+1}\}$ and $t_{i+1} = t_i$ (See figure 5d). To prove the satisfiability of the invariant we must first note that for this case to hold $C_i$ must have had more than one edge. If it had one edge then we would have been in the above case, not here. Therefore, $C_i$ has more than two edges. We need to prove that all the edges of $C_i$ satisfy the invariant since they are no longer in the clique. Firstly, the edge $e_{ii}$ does not intersect $e_{i+1}$ and hence satisfies the invariant. For the other edges of $C_i$, $e_{ii}$ acts as the $e_p$ given in the second case of the invariant. Hence the invariant holds after $i + 1$ steps.

(c) $R(e_{ij}) > R(e_{i+1})$. Set $C_{i+1} = \{e_{i+1}\}$ and $t_{i+1} = e_{ij}$ (See figure 5e). Again, we need to prove that all the edges of $C_i$ satisfy the invariant since they are no longer in the clique. This is trivially satisfied since all the edges of $C_i$ either intersect $t_{i+1}$ or are $t_{i+1}$. In each case, the invariant is satisfied.
14.3 Searching the Happy Clique

Now that we have a happy clique, we just need to find the vertex in it of the highest unoriented degree in order to find a safe reversal. Let \( C = \{e_1, \ldots, e_j\} \) be the happy clique with edges written in increasing left endpoint order. Since a clique is completely connected, we have \( L(e_1) < L(e_2) < \ldots < L(e_j) < R(e_1) < R(e_2) < \ldots < R(e_j) \). The following method is a linear time algorithm for finding the unoriented degree of each edge in \( C \). Let \( L(i) \) denote \( L(e_i) \). The edges partition the real line into disjoint intervals \( I_0, \ldots, I_{2j} \), where \( I_0 = (-\infty, L(1)] \), \( I_l = (L(l), L(l + 1)] \) for \( 1 \leq l < j \), \( I_j = (L(j), R(1)] \), \( I_l = (R(l - j), R(l - j + 1)] \) for \( j < l < 2j \) and \( I_{2j} = (R(j), \infty) \). The algorithm consists of following three stages.

1. Label each endpoint of unoriented vertices by the interval \( I_l \) they lie in.

2. Let \( A \) be an array of \( j \) counters. The algorithm will assign these counter so that the unoriented degree of \( e_l \) is \( \sum_{i=1}^{l} A[i] \). Atmost four counters are changed for each unoriented edge. This is done by traversing the permutation and if an unoriented gray edge is found, look at its endpoints and figure out which all counters to increment/decrement. The details can be seen from the paper [2].

3. Compute \( f = \max_l \{\sum_{i=1}^{l} A[i] \mid 1 \leq l \leq j\} \) and return \( e_f \).

14.4 Algorithm and Running Time

Since all of the above stages can be performed in linear time, searching the happy clique is linear time. Also finding a happy clique is linear time since at each vertex (gray edge) we spend only a constant amount of time.

The algorithm to optimally sort sort a permutation \( \pi \) is as follows,
1. Remove all hurdles from the permutation - $O(n\alpha(n))$ time. This algorithm is given in [3].

2. while the permutation is not sorted

   (a) Find a happy clique - $O(n)$ time.
   (b) Search the happy clique to find a reversal - $O(n)$ time
   (c) Carry out the reversal on $\pi$ and reconstruct $\pi^{-1}$ - $O(n)$ time.

The algorithm clearly runs in $O(r'n + n\alpha(n))$ time as we said earlier.

15 Algorithm given by [ANB]

This algorithm is based on finding a safe reversal by finding the edge with the maximum score. Here score is defined as $T + U - O + 1$ where $T$ is the number of oriented vertices, $U$ is the number of unoriented neighbours and $O$ is the number of oriented neighbours. Note that though we defined the score differently, we can easily adapt Theorem 10.4 to prove that such a reversal is also safe. In order to find the maximum score, we only need to look at $U - O$ for each edge. Although this idea is simpler than the above by [KST], it leads to higher time complexities. As we shall see later, this is because of lack of the clique structure.

The paper presents an algorithm which works on the adjacency matrix of $OV(\pi)$. The actual idea of the algorithm is very simple. First, the hurdles are removed using the same procedure as used above. Now we keep on finding a safe reversal and apply it to the graph $OV(\pi)$. Using the adjacency matrix, we can easily find out $U - O$ for each vertex of $OV(\pi)$ by looking at each edge once. Hence in $O(n^2)$ time, we can find a safe reversal. Since
we know how to update $OV(\pi)$ with each reversal, we can carry out the reversal on the graph in time $O(n^2)$. The paper[1] gives the implementation of this idea which takes $O(n) \ n$ – bit vector operations per reversal.

16 Algorithm given by Us

16.1 The Algorithm

We know that carrying out reversals on the graph $OV(\pi)$ can be $O(n^2)$ in the worst case. Since the operation involves the process of complementing a subgraph, the edges in $OV(\pi)$ keep on fluctuating. This prevents us from predicting how the graph will look after certain number of reversals. Therefore, the only way out is to use the implicit representation of $OV(\pi)$ by keeping the permutations $\pi$ and $\pi^{-1}$. This way we can reduce the time required to carry out a reversal to $O(n)$. Next we describe a procedure to find out the maximum score $(= U - O)$ edge in time $O(n \ log(n))$.

First we need a search tree that can support the following operations,

- Insert an element into the tree and also return the number of elements smaller than it in total $O(log(n))$ time.

- Delete an element from the tree in $O(log(n))$.

This data structure can be made by adapting any bounded height binary tree (like Red-Black trees, 2-3 trees) to maintain, at each node inside the tree, the number of elements in each of its subtrees. This information can be maintained at no extra cost.

The algorithm makes four passes of the permutation and in the first two passes it calculates the value of $U$ for each vertex. In the next two passes, it calculates $O$ for each vertex. Since
calculating $U$ is no different from calculating $O$, we will just describe the procedure to find $U$.

**A Pass:** The algorithm will traverse the permutation from left to right and at each step it maintains the following,

- $k =$ the number of unoriented edges whose left endpoint have been seen but not the right endpoints.
- $T =$ the tree containing the right endpoints of the above edges.

This information is updated and used in the following manner. Initialise $k = 0$ and $T = \emptyset$. If we are currently at $\pi_i$ then,

1. If there is no gray edge going out of $\pi_i$ then move on.

2. If this is the left endpoint of an unoriented edge $e$ then increment $k$ and insert $R(e)$ into $T$. The value returned by this insertion, which is the number of elements in $T$ which are smaller than $R(e)$, is the number of gray edges that start before $e$ and intersect it. The reason for this is clear from the fact that intersecting edges contain only one endpoint of the other.

3. If this is the right endpoint of an unoriented edge $e$ then delete $R(e)$ from $T$, decrement $k$ and move on.

4. If this is the left endpoint of an oriented gray edge $e$ then insert $R(e)$ into $T$. The value returned by this insertion, is the number of unoriented gray edge that start before $e$ and intersect it. Now delete $R(e)$ from $T$ to maintain the invariant ($e$ is oriented).

5. If this is the right endpoint of an oriented gray edge $e$ then move on.
This pass will give us, for each gray edge, the number of (un-oriented) gray edges that start before it and intersect it. The second pass will go from right to left, reversing the roles of right and left endpoints. The result of the second pass will thus be, for each edge, the number of (unoriented) gray edges that end to its right and intersect it. The sum of these values gives us $U$. To get $O$ for each edge, reverse the roles of oriented and unoriented edges and do the same two passes as above. Each pass clearly runs in $O(n \log(n))$ time since we spend $O(\log(n))$ time at each vertex. Therefore, we can find the score for each vertex in $O(n \log(n))$ time and thus find the reversal by finding the vertex with the maximum score in the same time. We can then carry out the reversal on $\pi$ and recompute $\pi^{-1}$ in linear time. Now we can repeat the process and optimally sort the permutation in $O(n \log(n))$ time per reversal.

16.2 An Unsolved Problem

The factor of $\log(n)$ can be removed from our algorithm if we can find a way of avoiding the tree $T$. Now, while traversing the permutation, we are maintaining the parameter $k$ which is the number of edges that start before the current position and have not yet finished. Only some of these edges will intersect the current edge being considered and others will pass over this edge i.e. they would have a span containing the span of the current edge. Therefore, if we can solve the following problem, we can find out the exact part of $k$ that intersects.

**Problem**: Given a set of $n$ intervals (gray edges) $[a_i, b_i]$ for $1 \leq i \leq n$ such that all of the numbers $\{a_1, a_2, \ldots, a_n, b_1, b_2, \ldots, b_n\}$ are distinct and lie between 1 and $2n$, find for each interval, the number of intervals that it contains (or is contained in).
Note that our algorithm solves a problem equivalent to the above in \( O(n \log(n)) \). We were not able to prove/disprove that this problem can be solved in linear time. We thus pose it as an open problem. If we do not restrict this containment problem to integers, but generalise it to reals, then the usual sorting of numbers trivially reduces to this problem in linear time and hence this general version of the problem has a complexity of atleast \( \Omega(n \log n) \). In our case, however, the numbers are integers and that too from a known set, so the time bound may be lower.

17 Acknowledgement

We would like to express sincere thanks to Prof. S.N. Maheshwari, Prof. Subhasis Banerjee and other members of our bioinformatics independent study group, who provided us valuable support and motivation to complete this study.

18 Bibliography


19 Appendix

Proof of Theorem 3.1: For the first part, see figure 6a. Since there is no gray edge incident on any of the four vertices, on application of the reversal, they will form the edges shown. If there is no black edge between $a$ and $b$ after the reversal, then there should have been a gray edge between them before the reversal as they differ only by one. This argument can be adapted to prove that the edges would be as shown. Clearly, here $\Delta b = 2$ and $\Delta c = 1$.

For the second part see figure 6b and note that the cycle $C$ would still connect $b$ and $c$ after the reversal. This gives rise to a large cycle but no extra cycle is added. Again $\Delta b = 1$ and $\Delta c = 0$ is obvious.

The third part (figure 6c) also follows the same argument as given in the above two cases. Here two different cycles $C_1$ and $C_2$ get merged into one cycle. Therefore, $\Delta b = 0$ and $\Delta c = -1$. One thing to note here is that the resulting merged cycle is oriented. To prove this, assume that the new cycle is unoriented. From the characterisation of oriented edges, we can see that for $(\pi_i, \pi_j)$ to be an unoriented gray edge, $i + j$ has to be odd. Now, because of the black edge $(a, b)$, $a$ must be even. Therefore any unoriented edge from $a$ must land up on an odd position say $a_2$. To continue along the cycle, we would have to take a black edge from $a_2$ to go to $a_3$ which would be even. Now I can take a grey edge from here to go to an odd position. Since I am traversing $C_1$ I should reach $b$ by a grey edge but this cannot be possible since all the grey edges land on odd positions and $b$ is even (due to the black edge $(b, d)$). This contradicts the fact that $C$ was unoriented. Therefore, $C$ is oriented.
The cases 4a and 4b are exactly the opposite of cases 1 and 2. The case 4d is exactly the opposite of case 3 as can be seen from the figure 7b. This leaves us with case 4c (figure 7a). The fact that both the paths $C_1$ and $C_2$ cannot be just one gray edge (neither $(b, d)$ nor $(a, c)$ is a gray edge) means that they would survive the reversal and still connect the same endpoints they used to connect before the reversal. The black edges also remain since $c$ differs from $a$ by atleast 2 (otherwise there would have been a gray edge $(a, c)$) and similarly $d$ must differ from $b$ by atleast 2. This means that $\Delta b = 0$ and $\Delta c = 0$. 


Figure 1: a) The breakpoint graph, $B(\pi)$, of the permutation $\pi = (4, -3, 1, -5, -2, 7, 6)$. Black edges are solid; gray edges are dashed; oriented edges are bold. b) $B(\pi)$ decomposes into two disjoint alternating cycles. c) The overlap graph, $OV(\pi)$. Black vertices correspond to oriented edges.
Oriented Edges

Unoriented Edges

Figure 2. The above graph shows how gray edges (dotted) look like w.r.t. to the black edges. Note that black edges only go from an even number to an odd number. This also shows that $i+j$ is even for an oriented gray edge going from the $i^{th}$ position of the permutation to the $j^{th}$ position.
This Figure shows how intersection of the edges e1 and e2 changes on applying the reversal r(e)

This Figure shows how the orientation of the edge f changes on applying the reversal r(e)

This Figure shows why oriented edges disappear on reversals. If there was no gray edge f then only e would disappear

Figure 3.
Figure 4. (a) shows a permutation having one maximum and one minimal hurdle. Figure (b) shows the permutation arranged on a circle. Note that the elements of the hurdles lie consecutively on the circle.
Figure: The various cases of the algorithm to find a happy clique. The topmost interval is always $t_i$. The three thick intervals comprise $C_i$. The dotted interval corresponds to $e_{i+1}$. 
Figure 6.
Figure 7.