GENOME REARRANGEMENTS

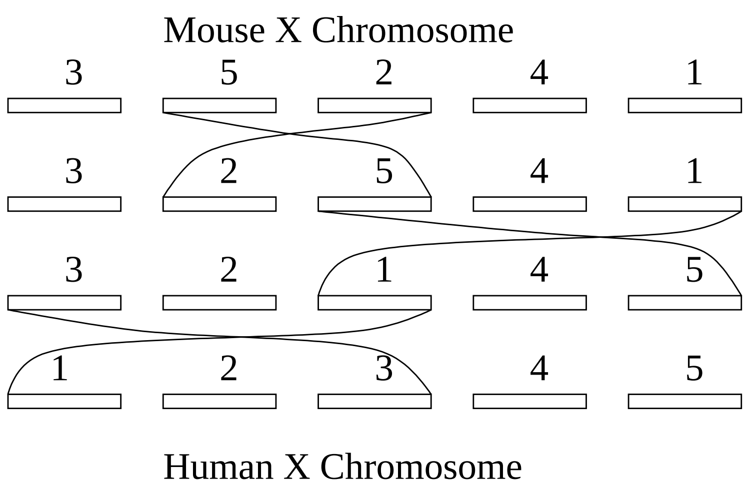
Computational Biology, Winter 2006

Lund University

Mia Persson

Biological Background

- Suppose that we want to compare entire genome across species.
- For example, we can compare the X chromosome of a mouse with the Human X chromosome, see Figure below.

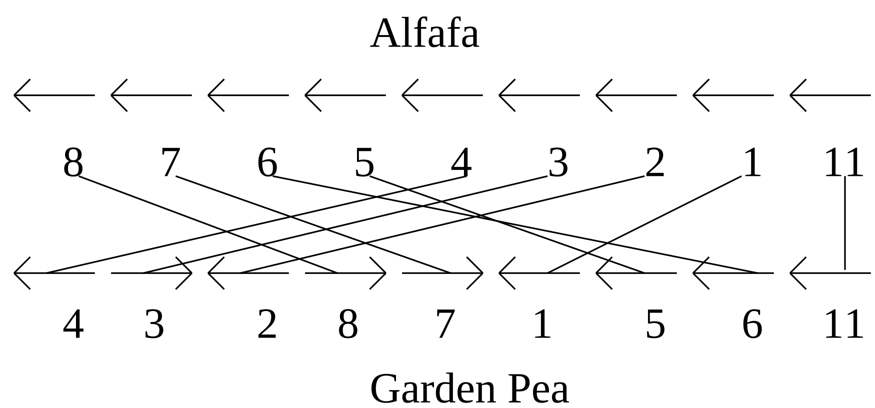


Biological Background

- Mutations where longer pieces of a chromosome are moved or copied to other location within the same chromosome or even to other chromosomes are called *genome rearrangements.*
- In their simplest form, rearrangement events can be modeled by a series of reversals that transform one genome into another, (see Human-Mouse example above).

Mathematical Model

- Consider the genome of two related species. We divide the genome into (possibly oriented) *blocks* where a block is a section of the genome containing one (or possibly more) gene (genes), see Figure below



- Two blocks in different genomes have the same value if they are homologous, that is, if they contain the same genes.

The Problem

- A *reversal operation* for a contiguous segments of oriented blocks is an operation that inverts the order of the affected blocks and also flip their arrows.

- Consider the following combinatorial optimization problem:

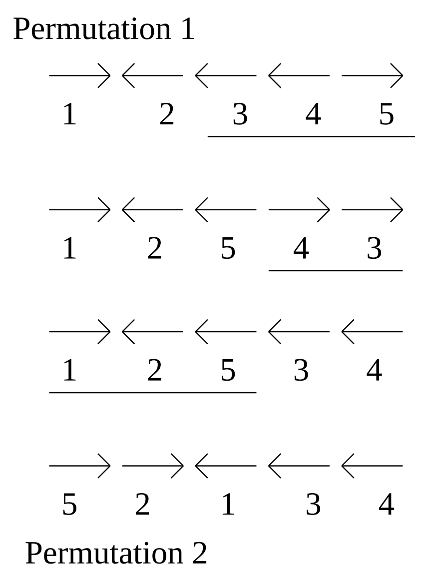
  Given two permutations of n oriented blocks originating from the chromosomes of two related organisms.

  Problem: Find the minimum number of reversals needed to transfer one permutation into the other.

- Here the minimum number of reversals comes from the assumption that Nature always finds paths that require a minimum number of changes (*the Parsimony assumption*).

The Problem - the Oriented Case

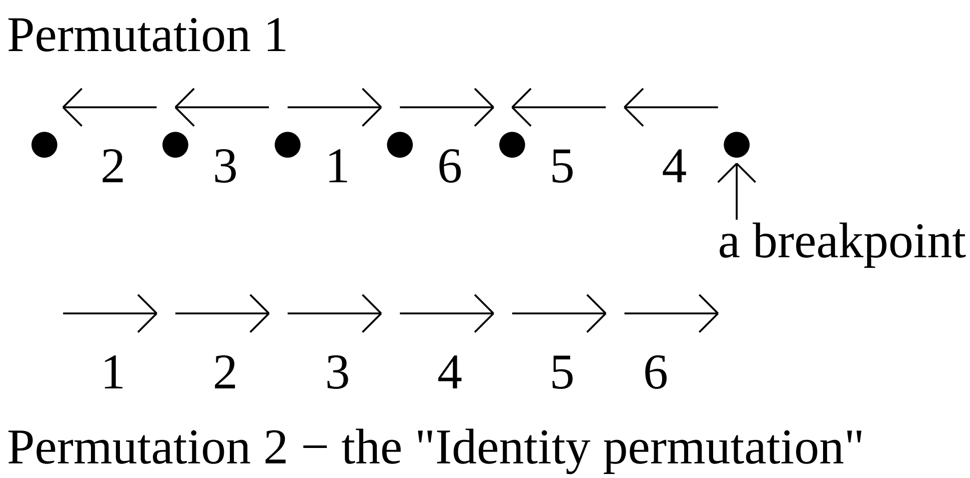
- Solvable in polynomial time.
- An example:



Distance is at least 3 reversals

Breakpoints

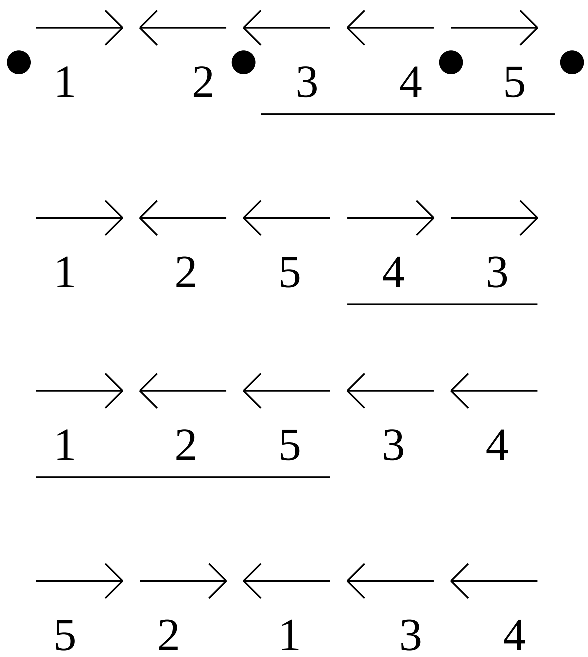
- **Definition:** A *breakpoint* is a point between two consecutive oriented labels that must be separated by at least one reversal.
- An example:



Breakpoints (cont' d)

- The *target permutation* has zero breakpoints by definition.
- A reversal can remove at most two breakpoints, because it cuts the permutation in exactly two locations.
- Hence, in the following example with four breakpoints, at least two reversals are needed to turn permutation 1 into permutation 2 (but in fact three reverals are needed).

Permutation 1



Permutation 2

Breakpoints (cont' d)

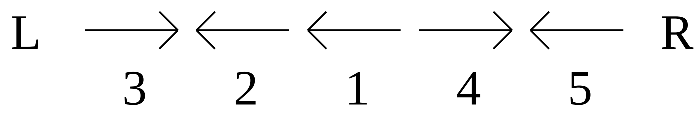
- **Definition**: Let $d(\alpha)$ denote the minimum number of reversals needed to bring the initial permutation α into the target permutation β. Let $b(\alpha)$ denote the number of breakpoints in α.
- $d(\alpha) \geq \frac{b(\alpha)}{2}$
- A reversal is *sorting* if it reduces the distance to the target permutation (by one).
- Note that a reversal can remove two endpoints without being sorting. (see exercise 5 in the coursebook).

The Diagram of Reality and Desire

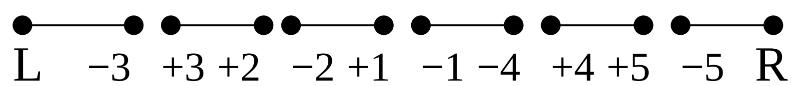
- The aforementioned lower bound $d(\alpha)$ is not very tight.
- We will now derive a better bound.
- In the following example we assume that the target permutation is the identity permutation.
- Some definitions...

The Diagram of Reality and Desire

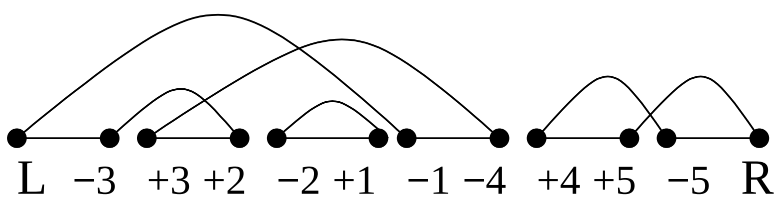
Permutation 1



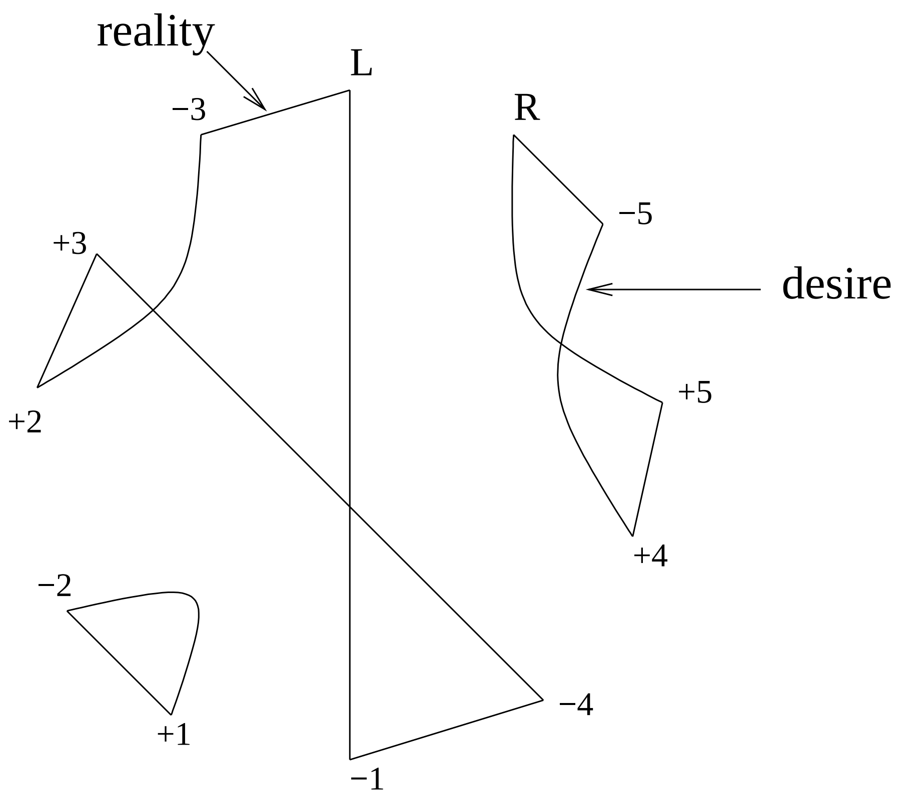
Reality Edges:



Reality and Desire Edges:



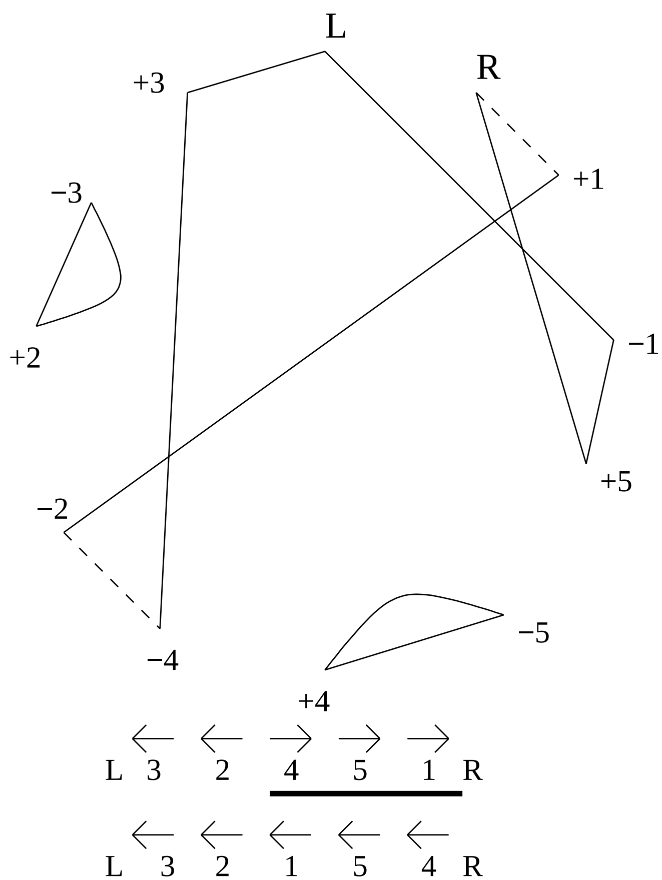
The Diagram of Reality and Desire



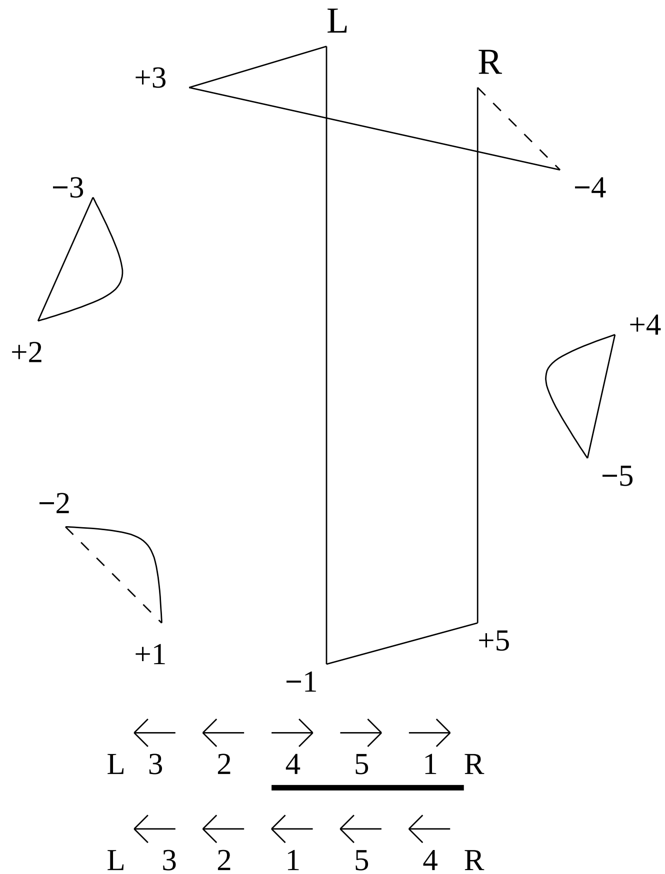
The Diagram of Reality and Desire

- Denote the Diagram (or graph) of Reality and Desire by $\mathrm{RD}(\alpha)$.
- Denote the number of cycles in $\mathrm{RD}(\alpha)$ by $c(\alpha)$.
- $c(\beta) = n + 1$, where n denotes the number of segments and β is the target permutation.
- The number of vertices in $\mathrm{RD}(\alpha)$ is $2n + 2$. This implies that there are $n + 1$ cycles in $\mathrm{RD}(\beta)$ (note that this is the permutation with the maximal number of cycles).
- Question: How does a reversal affect the cycles in $\mathrm{RD}(\alpha)$? Consider the following example:

Before the reversal:



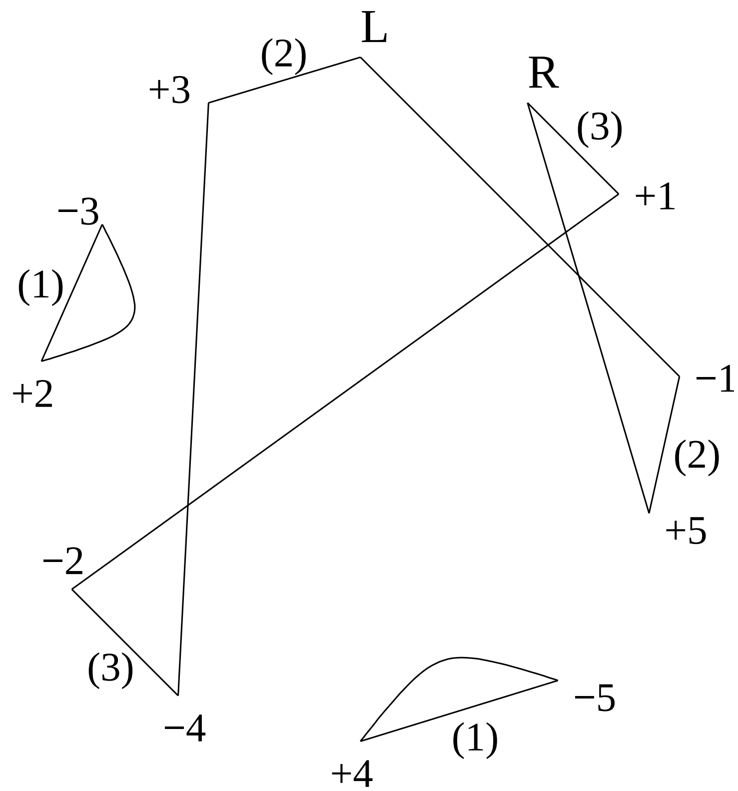
After the reversal:



Number of Cycles

Let $c(\alpha) =$ denote the number of cycles. $c(\Pi) = n - 1$ if Π is the identity permutation.

1. Reversal defined by two reality edges from different cycles decreases the number of cycles by one.
2. Converging edges from the same cycle does not increase the number of cycles.
3. Diverging edges from the same cycle increase the number of cycles by one.


L
R
(2)
+3
(3)
+1
−3
(1)
+2
−1
(2)
+5
−2
(3)
−4
−5
(1)
+4

A Better Lower Bound on $d(\alpha)$

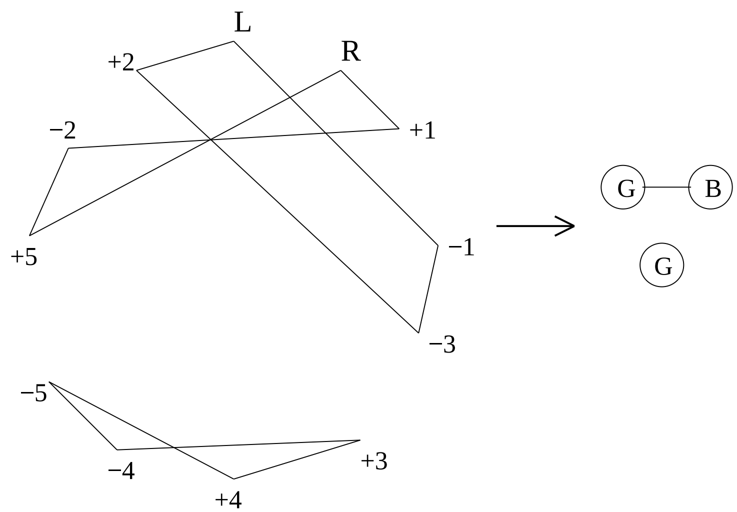
- $d(\alpha) \geq n + 1 - c(\alpha)$
- This lower is very good, but...
- It does not always work.

Good/Bad Cycles and Interleaving Graph

- The cycles in $RD(\alpha)$ can be classified as *good* or *bad*.
- A cycle is good if it has diverging edges, otherwise it is bad.
- Two cycles *interleave* if any pair of edges cross.

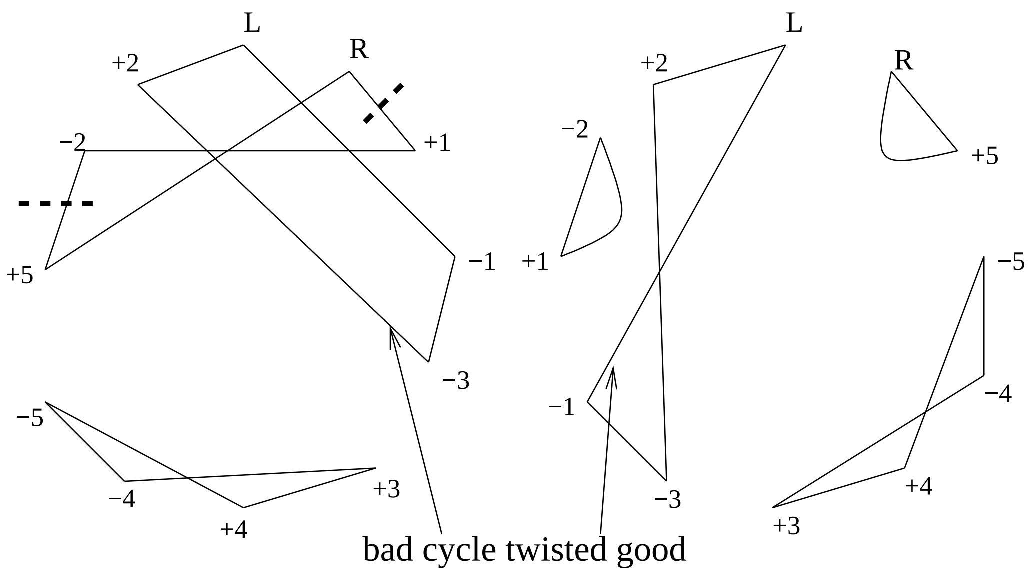
Good/Bad Cycles and Interleaving Graph

- The interleaving graph has cycles as nodes. Two nodes are connected if corresponding cycles interleave, but cycles of length 2 are excluded.
- A connected component of the interleaving graph is good if it contains at least one good cycle, otherwise it is bad.
- An example:



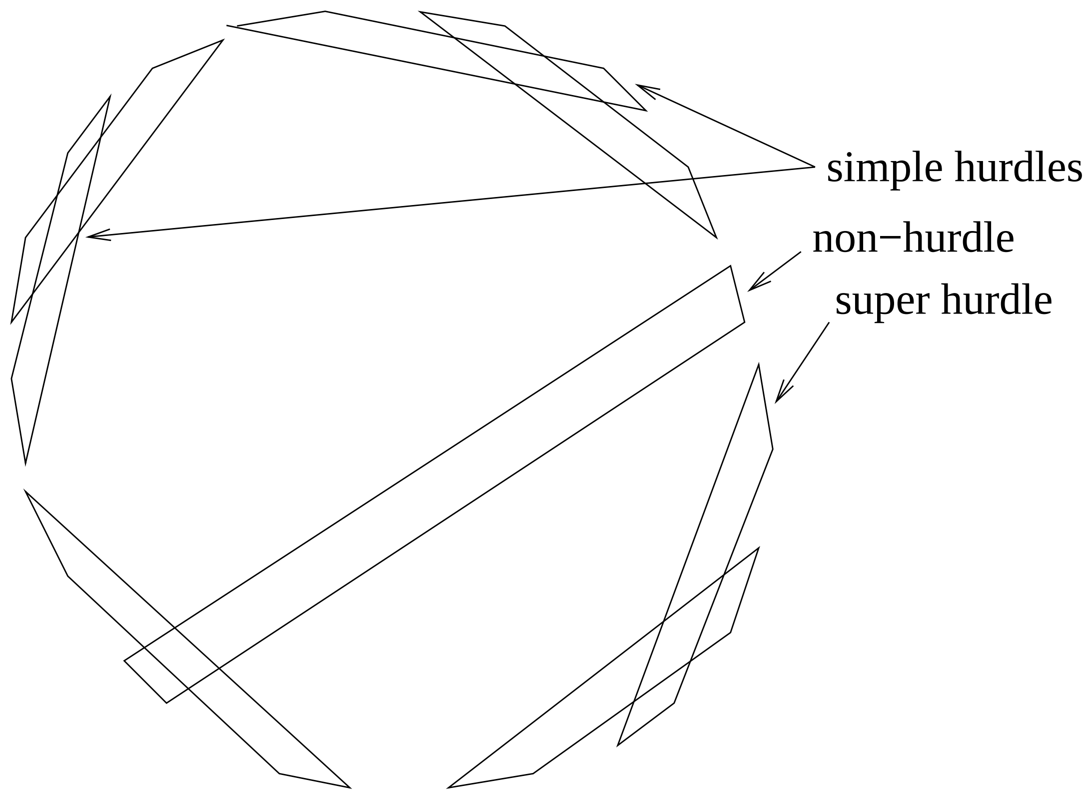
Good Components

- A reversal defined by two diverging edges of a good cycle is a sorting reversal if and only if its application does not lead to the creation of any bad components.
- As long as we have a good cycle, there will always be a sorting reversal of the kind that increases the number of cycles.



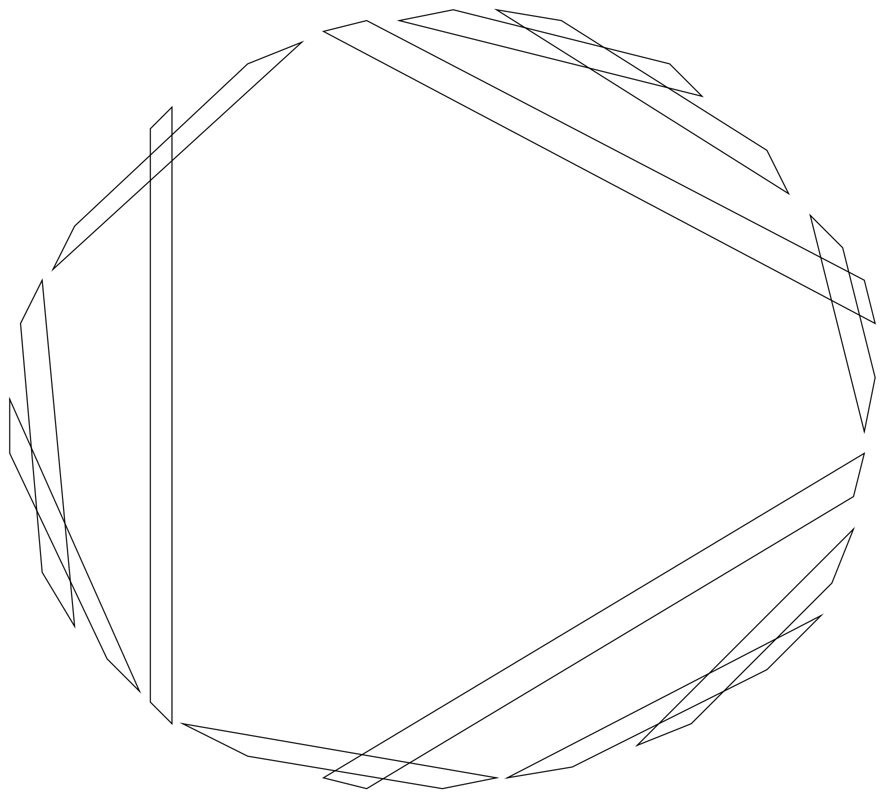
Bad Components and Hurdles

- A component B *separates* components A and C if every edge between A and C has to cross an edge of B.
- A *hurdle* is a bad component that does not separate any other two bad components, the other bad components are non-hurdles.
- A hurdle is a *super-hurdle* if its removal would cause a non-hurdle to become a hurdle.
- All other hurdles are called *simple hurdles*.


simple hurdles
non−hurdle
super hurdle

Fortress

- A *fortress* is a permutation which contains an odd number of hurdles and all of them are super hurdles.



Lower bound on $d(\alpha)$

- $d(\alpha) = n + 1 - c(\alpha) + h(\alpha) + f(\alpha)$
- $c(\alpha)$ is the number of cycles.
- $h(\alpha)$ is the number of hurdles.
- $f(\alpha)$ is 1 if α is a fortress and 0 otherwise.

Algorithm - Sorting Oriented Permutation

Pick a sorting reversal and perform it until target permutation is reached.

- If a good cycle exist pick a pair of diverging edges making sure that the corresponding reversal does not create any bad components
- If $h(\alpha)$ is odd and there is a simple hurdle, cut this hurdle (decreases number of hurdles by one without creating a fortress as $h(\alpha)$ is odd).
- If $h(\alpha)$ is odd and no simple hurdle exists, then α is a fortress, merge any two hurdles (either two less hurdles and one less cycle or fortress gone and one less hurdle.

Algorithm - Sorting Oriented Permutation (cont' d)

- If $h(\alpha)$ is even then merge two opposite hurdles. Choosing opposite hurdles will not create a new hurdle (two less hurdless)

The Problem - The Unoriented Case

- Given two permutations n blocks originating from the chromosomes of two related organisms.

  Problem: Find the minimum number of reversals needed to transfer one permutation into the other.

The Problem - The Unoriented Case

- NP-hard
- An example:

Permutation 1

1 2 3 4 5

1 2 5 4 3

1 2 5 3 4

5 2 1 3 4

Permutation 2

The Unoriented Case - Strips

- A breakpoint exists between every pair of non-consecutive label.
- A sequence of consecutive labels surrounded by breakpoints is a *strip*.
- Strips are either *increasing*, *decreasing* or both.
- L and R is always part of a single increasing strip.

An Example

L 1 2 | 8 7 | 3 | 5 6 | 4 R

Increasing strips: RL12, 56

Decreasing strip: 87

Both: 3, 4

Decreasing Strips

- If a permutation contains a decreasing strip, then it is always possible to decrease the number of breakpoints.
- A permutation always contains at least one increasing strip (RL).
- Pick the lowest label k in a decreasing strip.

$$\ldots \quad k-2 \quad k-1 \,\Big|\, \ldots \quad k+1 \quad k \,\Big|\, \ldots$$

$$\ldots \quad k-2 \quad k-1 \quad k \quad k+1 \quad \ldots \,\Big|\, \ldots$$

or

$$\ldots \quad k+1 \quad k \,\Big|\, \ldots \; k-2 \;\; k-1 \,\Big|\, \ldots$$

$$\ldots \quad k+1 \quad k \quad k-1 \;\; k-2 \ldots \,\Big|\, \ldots$$

Algorithm

- If a label k belongs to a decreasing strip and $k-1$ belongs to an increasing strip, then there is a reversal that removes at least one breakpoint.
- If label k belongs to a decreasing strip and $k+1$ belongs to an increasing strip, then there is a reversal that removes at least one breakpoint.
- Let α be a permutation with a decreasing strip. If all revesals that remove breakpoints from α leave no decreasing strip, then there is a revesal that removes two breakpoints from α.
- If there are no decreasing strips, do any reversal that cuts two breakpoints.

Algorithm

Until done:

- Apply reversals to decreasing strips with the smallest possible label provided that the resulting permutation has a decreasing strip.
- If the resulting permutation does not have a decreasing strips, do any reversal that cuts two breakpoints.

Note that as long as we have decreasing strips, we can always reduce the number of breakpoints with at least one.

Algorithm Analysis

- For every reversal (but the first) that does not decrease the number of breakpoints, the previous one decreased it by two.
- The last reversal decreases the number of breakpoints by two.
- Thus the number rate of breakpoints reductions is not less than half the optimum and we have a 2-approximation.