Greedy Algorithms

CS 498 SS

Saurabh Sinha

Chapter 5.5

A greedy approach to the motif finding problem

- Given t sequences of length n each, to find a profile matrix of length l.
- Enumerative approach $O(l\, n^t)$
  - Impractical
- Instead consider a more practical algorithm called “GREEDYMOTIFSEARCH”

Greedy Motif Search

- Find two closest l-mers in sequences 1 and 2 and form
  2 x l alignment matrix with *Score(***s***,2,DNA)*
- At each of the following *t-2* iterations, finds a "best" *l*-mer in sequence *i* from the perspective of the already constructed *(i-1) x l* alignment matrix for the first *(i-1)* sequences
- In other words, it finds an *l*-mer in sequence *i* maximizing

  *Score(***s***,i,DNA)*

  under the assumption that the first *(i-1) l*-mers have been already chosen
- Sacrifices optimal solution for speed: in fact the bulk of the time is actually spent locating the first 2 *l*-mers

Greedy Motif Search pseudocode

- GREEDYMOTIFSEARCH (**DNA**, t, n, l)
- **bestMotif** := (1,…,1)
- s := (1,…,1)
- for s_1=1 to n-l+1
  for s_2 = 1 to n-l+1
  if (**Score**(s,2,DNA) > **Score(bestMotif,**2,DNA)
  bestMotif$_1$:= s_1
  bestMotif$_2$:= s_2
- s_1 := **bestMotif**$_1$; s_2 := **bestMotif**$_2$
- for i = 3 to t
  for s_i = 1 to n-l+1
  if (**Score**(s,i,DNA) > **Score(bestMotif,**i,DNA)
  bestMotif$_i$:= s_i
  s_i := **bestMotif**$_i$
- Return **bestMotif**

A digression

- Score of a profile matrix looks only at the “majority” base in each column, not at the entire distribution
- The issue of non-uniform “background” frequencies of bases in the genome
- A better “score” of a profile matrix ?

Information Content

- First convert a “profile matrix” to a “position weight matrix” or PWM
  - Convert frequencies to probabilities
- PWM W: $W_{\beta k}$ = frequency of base β at position k
- q_β = frequency of base β by chance
- Information content of W:

$$\sum_k \sum_{\beta \in \{A,C,G,T\}} W_{\beta k} \log \frac{W_{\beta k}}{q_\beta}$$

Information Content

- If $W_{\beta k}$ is always equal to q_{β}, i.e., if W is similar to random sequence, information content of W is 0.
- If W is different from q, information content is high.

Greedy Motif Search

- Can be trivially modified to use “Information Content” as the score
- Use statistical criteria to evaluate significance of Information Content
- At each step, instead of choosing the top (1) partial motif, keep the top k partial motifs
  - “Beam search”
- The program “CONSENSUS” from Stormo lab.

- Further Reading: Hertz, Hartzell & Stormo, CABIOS (1990)
  http://bioinf.kvl.dk/~gorodkin/teach/bioinf2004/hertz90.pdf

Genome Rearrangements

Genome Rearrangements

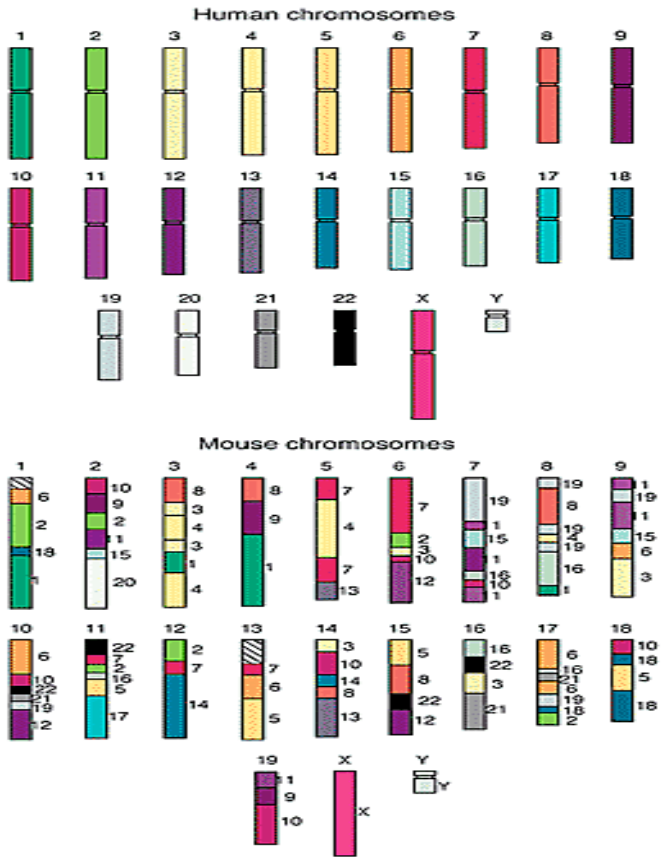
- Most mouse genes have human orthologs (i.e., share common evolutionary ancestor)
- The sequence of genes in the mouse genome is not exactly the same as in human
- However, there are subsets of genes with preserved order between human-mouse (“in synteny”)

Genome Rearrangements

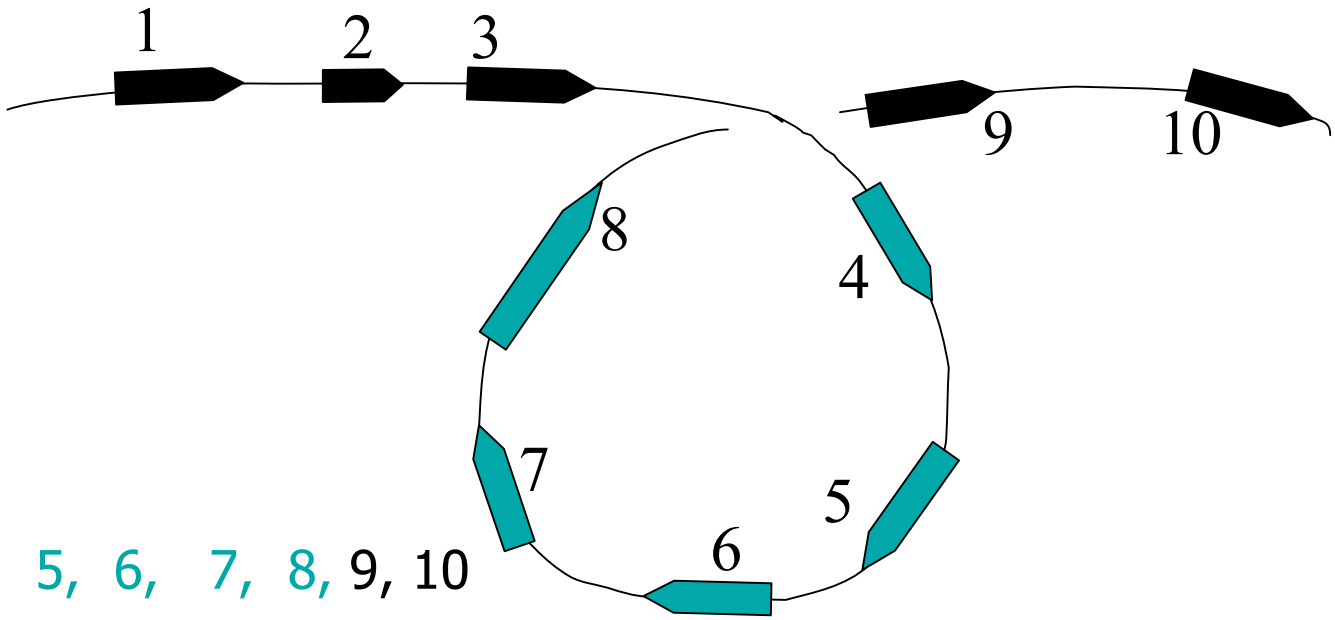
- The mouse genome can be cut into ~300 (not necessarily equal) pieces and joined pasted together in a different order (“rearranged”) to obtain the gene order of the human genome
- Synteny blocks
- Synteny blocks from different chromosomes in mouse are together on the same chromosome in human

Comparative Genomic Architectures: Mouse vs Human Genome

- Humans and mice have similar genomes, but their genes are ordered differently
- ~245 rearrangements
  - Reversals
  - Fusions
  - Fissions
  - Translocation

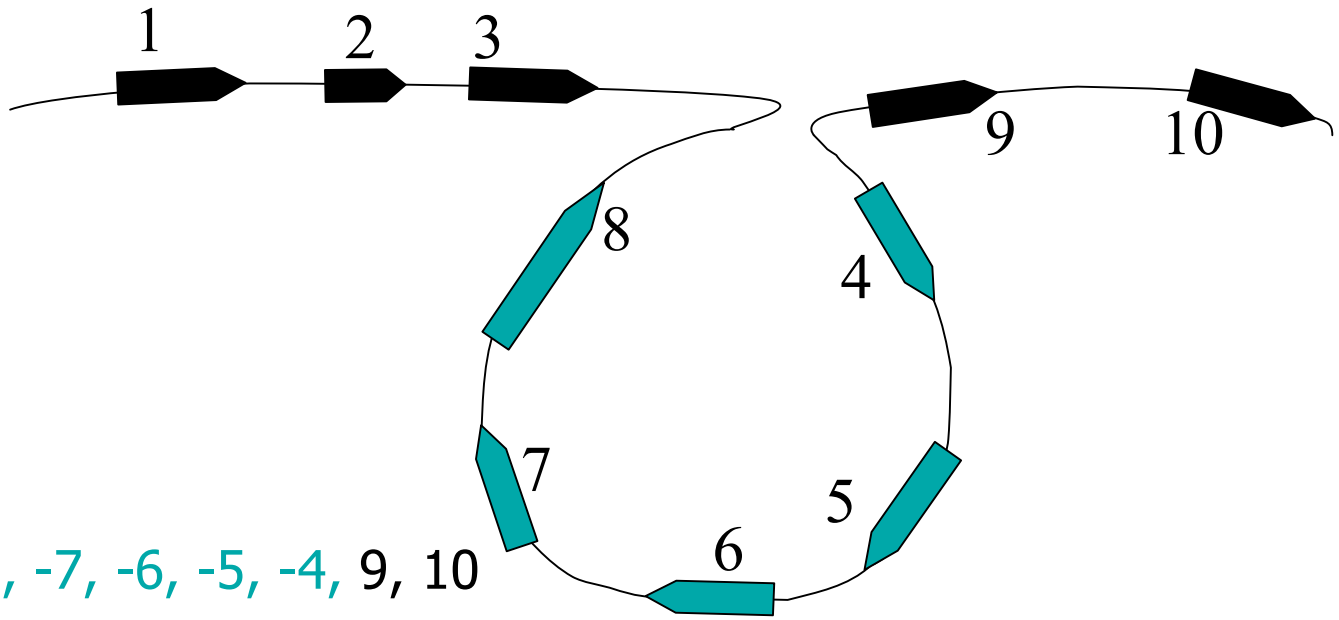


A type of rearrangement: Reversal



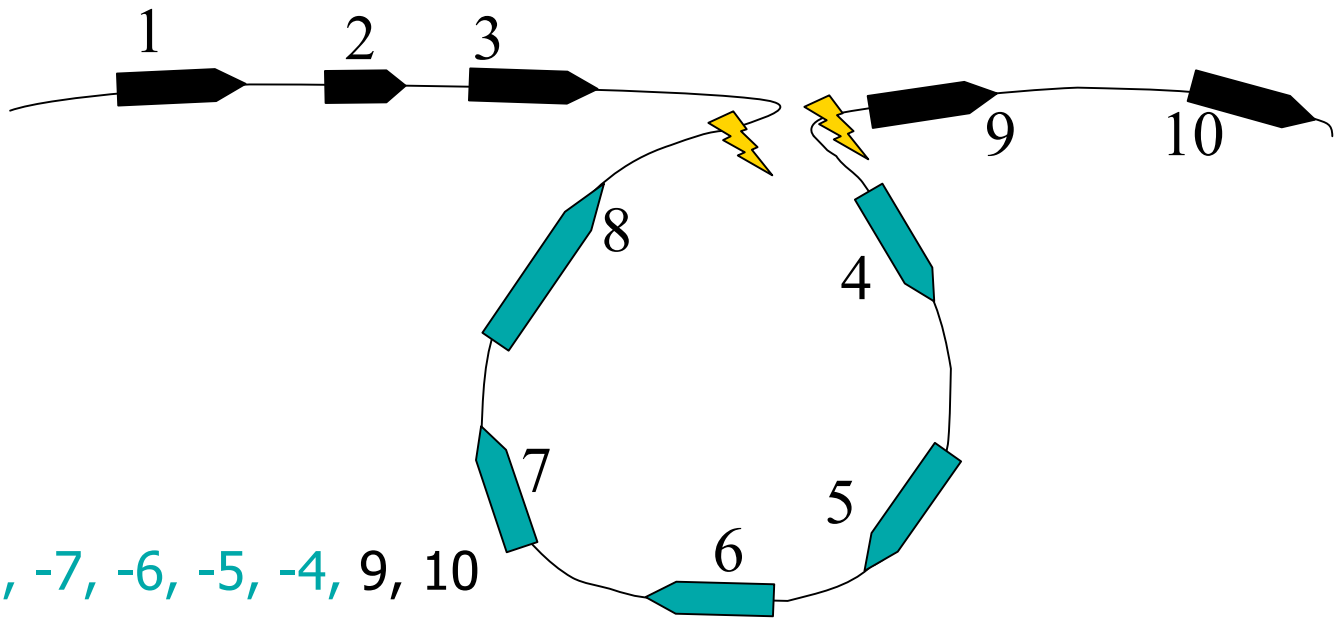
1, 2, 3, 4, 5, 6, 7, 8, 9, 10

A type of rearrangement: Reversal



1, 2, 3, -8, -7, -6, -5, -4, 9, 10

Breakpoints

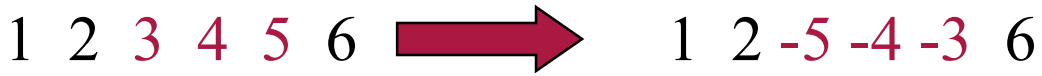


1, 2, 3, -8, -7, -6, -5, -4, 9, 10

The reversal introduced two breakpoints

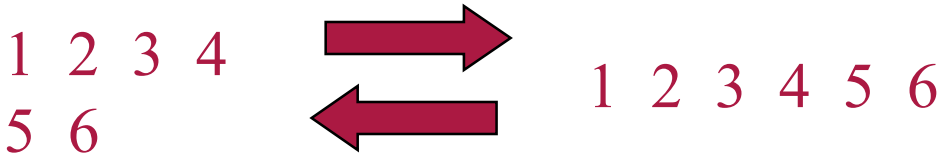
Types of Rearrangements

Reversal



Translocation

Fusion

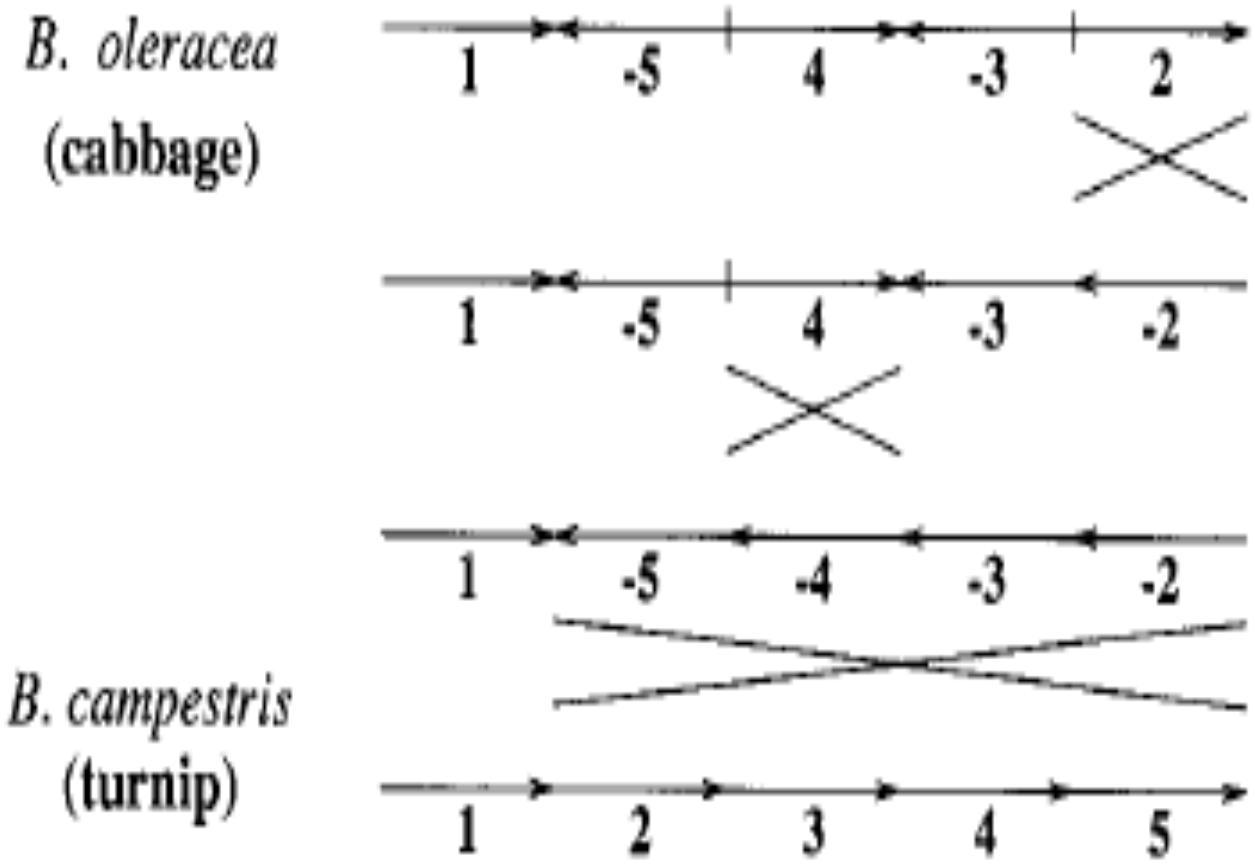


Fission

Turnip vs Cabbage: Almost Identical mtDNA gene sequences

- In 1980s Jeffrey Palmer studied evolution of plant organelles by comparing mitochondrial genomes of the cabbage and turnip
- 99% similarity between genes
- These surprisingly identical gene sequences differed in gene order
- This study helped pave the way to analyzing genome rearrangements in molecular evolution

Transforming Cabbage into Turnip



Genome Rearrangement

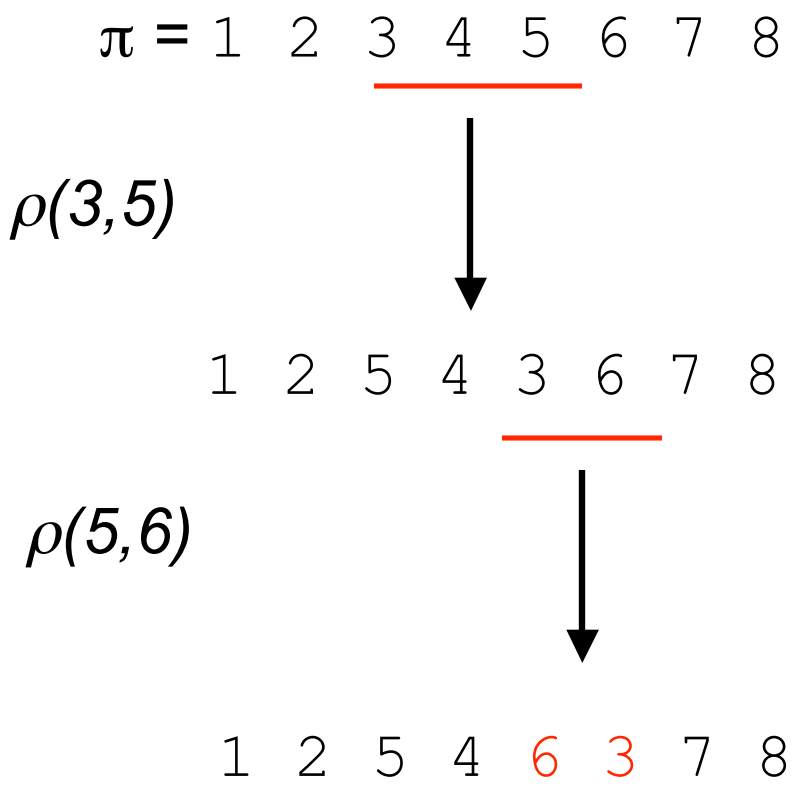
- Consider reversals only.
  - These are most common
- How to transform one genome (i.e., gene ordering) to another, using the least number of reversals ?

Reversals: Example

π = 1 2 3 4 5 6 7 8

1 2 5 4 3 6 7 8

Reversals: Example



Reversals and Gene Orders

- Gene order is represented by a permutation π:

$$\pi = \pi_1 \text{ ----- } \pi_{i-1}\ \pi_i\ \pi_{i+1} \text{ ----- } \pi_{j-1}\ \pi_j\ \pi_{j+1} \text{ ----- } \pi_n$$

$$\rho(i,j)$$

$$\pi_1 \text{ ----- } \pi_{i-1}\ \pi_j\ \pi_{j-1} \text{ ----- } \pi_{i+1}\ \pi_i\ \pi_{j+1} \text{ ----- } \pi_n$$

- Reversal $\rho(i, j)$ reverses (flips) the elements from i to j in π

Reversal Distance Problem

- Goal: Given two permutations, find the shortest series of reversals that transforms one into another
- Input: Permutations π and σ
- Output: A series of reversals $\rho_1,\ldots\rho_t$ transforming π into σ, such that t is minimum
- $\boldsymbol{t}$ - reversal distance between π and σ

Sorting By Reversals Problem

- Goal: Given a permutation, find a shortest series of reversals that transforms it into the identity permutation (*1 2 … n*)

- Input: Permutation π

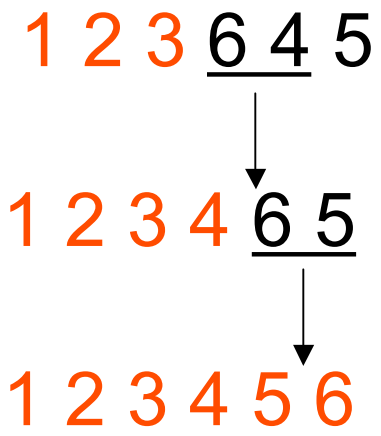
- Output: A series of reversals $\rho_1, \ldots \rho_t$ transforming π into the identity permutation such that *t* is minimum

Sorting By Reversals: A Greedy Algorithm

- If sorting permutation π = 1 2 3 6 4 5, the first three elements are already in order so it does not make any sense to break them.
- The length of the already sorted prefix of π is denoted *prefix*(π)
  - *prefix*(π) = 3
- This results in an idea for a greedy algorithm: increase *prefix*(π) at every step

Greedy Algorithm: An Example

- Doing so, π can be sorted



- Number of steps to sort permutation of length *n* is at most *(n – 1)*

Greedy Algorithm: Pseudocode

SimpleReversalSort(π)

1 **for** $i \leftarrow 1$ to $n - 1$

2 $j \leftarrow$ position of element i in π (i.e., $\pi_j = i$)

3 **if** $j \neq i$

4 $\pi \leftarrow \pi * \rho(i, j)$

5 **output** π

6 **if** π is the identity permutation

7 **return**

Analyzing SimpleReversalSort

- SimpleReversalSort does not guarantee the smallest number of reversals and takes five steps on π = 6 1 2 3 4 5 :

  - Step 1: 1 6 2 3 4 5
  - Step 2: 1 2 6 3 4 5
  - Step 3: 1 2 3 6 4 5
  - Step 4: 1 2 3 4 6 5
  - Step 5: 1 2 3 4 5 6

Analyzing SimpleReversalSort (cont'd)

- But it can be sorted in two steps:

  π = 6 1 2 3 4 5

  – Step 1: 5 4 3 2 1 6

  – Step 2: 1 2 3 4 5 6

- So, SimpleReversalSort(π) is not optimal

- Optimal algorithms are unknown for many problems; approximation algorithms are used

Approximation Algorithms

- These algorithms find approximate solutions rather than optimal solutions
- The approximation ratio of an algorithm A on input π is:

$$A(\pi) / OPT(\pi)$$

where

$A(\pi)$ -solution produced by algorithm A

$OPT(\pi)$ - optimal solution of the problem

Approximation Ratio/Performance Guarantee

- Approximation ratio (performance guarantee) of algorithm A: max approximation ratio of all inputs of size *n*
- For algorithm A that minimizes objective function (minimization algorithm):
  - $\max_{|\pi| = n} A(\pi) / OPT(\pi)$
- For maximization algorithm:
  - $\min_{|\pi| = n} A(\pi) / OPT(\pi)$

Adjacencies and Breakpoints

$$\pi = \pi_1\pi_2\pi_3\ldots\pi_{n-1}\pi_n$$

- A pair of elements π_i and π_{i+1} are adjacent if

$$\pi_{i+1} = \pi_i \pm 1$$

- For example:

π = 1 9 3 4 7 8 2 6 5

- (3, 4) or (7, 8) and (6,5) are adjacent pairs

Breakpoints: An Example

There is a breakpoint between any pair of adjacent elements that are non-consecutive:

$$\pi = 1 \,|\, 9 \,|\, 3 \;\; 4 \,|\, 7 \;\; 8 \,|\, 2 \,|\, 6 \;\; 5$$

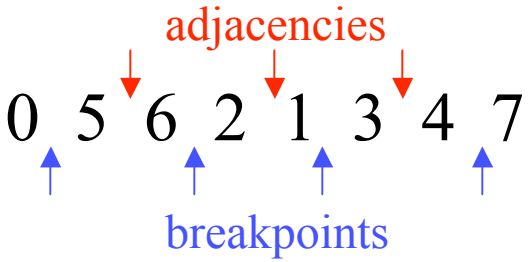
- Pairs (1,9), (9,3), (4,7), (8,2) and (2,6) form breakpoints of permutation π
- $b(\pi)$ - # breakpoints in permutation π

Adjacency & Breakpoints

- An adjacency - a pair of adjacent elements that are consecutive
- A breakpoint - a pair of adjacent elements that are not consecutive

---

$\pi = 5\ \ 6\ \ 2\ \ 1\ \ 3\ \ 4$ ⟶ Extend π with $\pi_0 = 0$ and $\pi_7 = 7$



Reversal Distance and Breakpoints

- Each reversal eliminates at most 2 breakpoints.

π = 2 3 1 4 6 5

0 | 2 3 | 1 | 4 | 6 5 | *7*	$b(\pi) = 5$
0 1 | 3 2 | 4 | 6 5 | *7*	$b(\pi) = 4$
0 1 2 3 4 | 6 5 | *7*	$b(\pi) = 2$
0 1 2 3 4 5 6 *7*	$b(\pi) = 0$

Reversal Distance and Breakpoints

- Each reversal eliminates at most 2 breakpoints.

reversal distance ≥ #breakpoints / 2

π = 2 3 1 4 6 5

0 | 2 3 | 1 | 4 | 6 5 | *7*	$b(\pi) = 5$
0 1 | 3 2 | 4 | 6 5 | *7*	$b(\pi) = 4$
0 1 2 3 4 | 6 5 | *7*	$b(\pi) = 2$
0 1 2 3 4 5 6 *7*	$b(\pi) = 0$

Sorting By Reversals: A Better Greedy Algorithm

BreakPointReversalSort(π)

1 **while** $b(\pi) > 0$

2 Among all possible reversals, choose reversal ρ minimizing $b(\pi \cdot \rho)$

3 $\pi \leftarrow \pi \cdot \rho(i, j)$

4 **output** π

5 **return**

Sorting By Reversals: A Better Greedy Algorithm

BreakPointReversalSort(π)

1 **while** $b(\pi) > 0$

2 Among all possible reversals, choose reversal ρ minimizing $b(\pi \cdot \rho)$

3 $\pi \leftarrow \pi \cdot \rho(i, j)$

4 **output** π

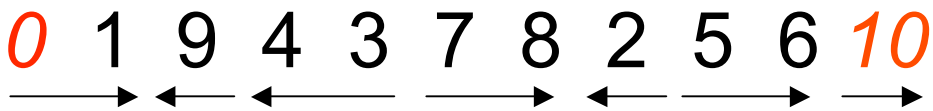
5 **return**

Thoughts on BreakPointReversalsSort

- A “different face of greed”: breakpoints as the marker of progress
- Why is this algorithm better than SimpleReversalSort ? Don’t know how many steps it may take
- Does this algorithm even terminate ?
- We need some analysis …

Strips

- Strip: an interval between two consecutive breakpoints in a permutation
  - Decreasing strip: *strip* of elements in decreasing order
  - Increasing strip: *strip* of elements in increasing order



  - A single-element strip can be declared either increasing or decreasing. We will choose to declare them as decreasing with exception of the strips with *0* and *n+1*

Reducing the Number of Breakpoints

Theorem 1:

If permutation π contains at least one decreasing strip, then there exists a reversal ρ which decreases the number of breakpoints (i.e. $b(\pi \bullet \rho) < b(\pi)$)

Things To Consider

- For π = 1 4 6 5 7 8 3 2

  0 1 | 4 | 6 5 | 7 8 | 3 2 | *9* $b(\pi) = 5$

  – Choose decreasing strip with the smallest element *k* in π (*k* = 2 in this case)

Things To Consider

- For π = 1 4 6 5 7 8 3 2

  0 1 | 4 | 6 5 | 7 8 | 3 2 | *9* $b(\pi) = 5$

  – Choose decreasing strip with the smallest element *k* in π (*k* = 2 in this case)

Things To Consider

- For π = 1 4 6 5 7 8 3 2

  0 1 | 4 | 6 5 | 7 8 | 3 2 | *9* $b(\pi) = 5$

  – Choose decreasing strip with the smallest element *k* in π (*k* = 2 in this case)

  – Find *k* – *1* in the permutation

Things To Consider

- For π = 1 4 6 5 7 8 3 2

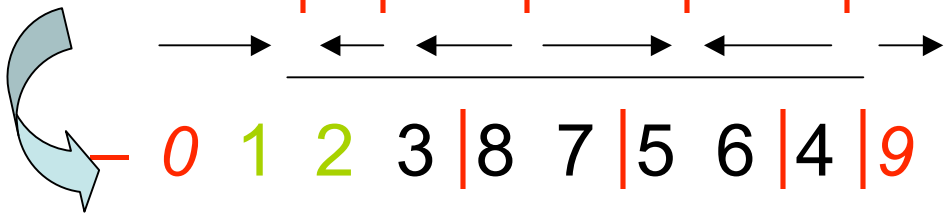
  0 1 | 4 | 6 5 | 7 8 | 3 2 | *9* $b(\pi) = 5$

  – Choose decreasing strip with the smallest element *k* in π (*k* = 2 in this case)

  – Find *k* – *1* in the permutation

  – Reverse the segment between *k* and *k-1*:

  – *0* 1 | 4 | 6 5 | 7 8 | 3 2 | *9* $b(\pi) = 5$



  – *0* 1 2 3 | 8 7 | 5 6 | 4 | *9* $b(\pi) = 4$

Reducing the Number of Breakpoints (Again)

- If there is no decreasing strip, there may be no reversal ρ that reduces the number of breakpoints (i.e. $b(\pi \bullet \rho) \geq b(\pi)$ for any reversal ρ).
- By reversing an increasing strip (# of breakpoints stay unchanged), we will create a decreasing strip at the next step. Then the number of breakpoints will be reduced in the next step (theorem 1).

ImprovedBreakpointReversalSort

ImprovedBreakpointReversalSort(π)

1 **while** $b(\pi) > 0$
2 **if** π has a decreasing strip
3 Among all possible reversals, choose reversal ρ

that minimizes $b(\pi \cdot \rho)$

4 **else**
5 Choose a reversal ρ that flips an increasing strip in π
6 $\pi \leftarrow \pi \cdot \rho$
7 **output** π
8 **return**

ImprovedBreakpointReversalSort: Performance Guarantee

- *ImprovedBreakPointReversalSort* is an approximation algorithm with a performance guarantee of at most 4
  - It eliminates at least one breakpoint in every two steps; at most $2b(\pi)$ steps
  - Approximation ratio: $2b(\pi) \ / \ d(\pi)$
  - Optimal algorithm eliminates at most 2 breakpoints in every step: $d(\pi) \geq b(\pi) / 2$
  - Performance guarantee:
    - $(\ 2b(\pi) / d(\pi) \ ) \geq [\ 2b(\pi) / (b(\pi) / 2) \ ] = \ 4$