Genome Rearrangements

Study Chapters 5.3-5.5

Python Tutorial on 9/11
From Last Time

- We developed a SimpleReversalSort algorithm that sorts by extending its prefix on every iteration (n-1) steps.
- On $\pi: 6 \underline{1} \underline{2} 3 4 5$
  
  Flip 1: $1 \underline{6} 2 3 4 5$
  Flip 2: $1 2 \underline{6} 3 4 5$
  Flip 3: $1 2 3 \underline{6} 4 5$
  Flip 4: $1 2 3 4 \underline{6} 5$
  Flip 5: $1 2 3 4 5 \underline{6}$

- But it could have been sorted in two flips:

  $\pi: \underline{6} 1 2 3 4 5$

  Flip 1: $5 \underline{4} 3 2 1 6$

  Flip 2: $1 2 3 4 5 6$

We probably don't want to use this algorithm to estimate the reversal distance between two genomes.
Approximation Algorithms

• Today’s algorithms find approximate solutions rather than optimal solutions.
• The approximation ratio of an algorithm $A$ on input $\pi$ is:
  $$\frac{A(\pi)}{OPT(\pi)}$$

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 $\mathcal{A}$: max approximation ratio over all inputs of size $n$
  
  - For a minimizing algorithm $\mathcal{A}$ (like ours):
    - Approx Ratio = $\max |\pi| = n \frac{\mathcal{A}(\pi)}{\text{OPT}(\pi)} \geq 1.0$

  - For maximization algorithms:
    - Approx Ratio = $\min |\pi| = n \frac{\mathcal{A}(\pi)}{\text{OPT}(\pi)} \leq 1.0$
Approximation Ratio

SimpleReversalSort(\(\pi\))
1 for \(i \leftarrow 1\) to \(n - 1\)
2 \(j \leftarrow\) position of element \(i\) in \(\pi\) (i.e., \(\pi_j = i\))
3 if \(j \neq i\)
4 \(\pi \leftarrow \pi \rho(i, j)\)
5 output \(\pi\)
6 if \(\pi\) is the identity permutation
7 return

Step 0: 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

Step 0: 6 1 2 3 4 5
Step 1: 5 4 3 2 1 6
Step 2: 1 2 3 4 5 6

A(\(\pi\))? n-1
OPT(\(\pi\))? approximation ratio?
any better greedy algorithms?
New Idea: Adjacencies

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

- A pair of neighboring elements \( \pi_i \) and \( \pi_{i+1} \) are adjacent if
\[ \pi_{i+1} = \pi_i \pm 1 \]

- For example:
\[ \pi = 1 \ 9 \ 3 \ 4 \ 7 \ 8 \ 2 \ 6 \ 5 \]

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

**Breakpoints** occur between neighboring non-adjacent elements:

\[ \pi = 1 \underline{9} 3 4 7 \underline{8} 2 6 5 \]

- Pairs (1,9), (9,3), (4,7), (8,2) and (2,5) define 5 breakpoints of permutation \( \pi \)

- \( b(\pi) \) - # breakpoints in permutation \( \pi \)
Extending Permutations

• One can place two elements \( \pi_0 = 0 \) and \( \pi_{n+1} = n+1 \) at the beginning and end of \( \pi \) respectively.

\[
\pi = 1 \ 9 \ 3 \ 4 \ 7 \ 8 \ 2 \ 6 \ 5
\]

Extending with 0 and 10

\[
\pi = 0 \ 1 \ 9 \ 3 \ 4 \ 7 \ 8 \ 2 \ 6 \ 5 \ 10
\]

A new breakpoint was created after extending.

An extended permutation of \( n \) can have at most \( (n+1) \) breakpoints, \( (n-1 \) between elements plus 2)
Breakpoints are the *bottlenecks* for sorting by reversals once they are removed, the permutation is sorted.

Each “*useful*” reversal eliminates at least 1 and at most 2 breakpoints.

Consider the following application of $\text{SimpleReversalSort}(\text{Extend}(\pi))$:

$$\pi = 2 \ 3 \ 1 \ 4 \ 6 \ 5$$

$$\begin{array}{ccccccc}
0 & 2 & 3 & 1 & 4 & 6 & 5 & 7 \\
\hline
0 & 1 & 3 & 2 & 4 & 6 & 5 & 7 \\
0 & 1 & 2 & 3 & 4 & 6 & 5 & 7 \\
0 & 1 & 2 & 3 & 4 & 5 & 6 & 7
\end{array}$$

$b(\pi) = 5$

$b(\pi) = 4$

$b(\pi) = 2$

$b(\pi) = 0$

$$\text{required reversals} \geq \frac{b(\pi)}{2}$$
Sorting By Reversals:
A Better Greedy Algorithm

BreakPointReversalSort(\(\pi\))

1. \textbf{while} \(b(\pi) > 0\)
2. Among all possible reversals, choose reversal \(\rho\) minimizing \(b(\pi \cdot \rho)\)
3. \(\pi \leftarrow \pi \cdot \rho(i, j)\)
4. output \(\pi\)
5. return

The "greedy" concept here is to reduce as many breakpoints as possible

Does it always terminate?

What if no reversal reduces the number of breakpoints?
New Concept: *Strips*

- **Strip**: an interval between two consecutive breakpoints in a permutation
  - **Decreasing strip**: strip of elements in decreasing order (e.g. 6 5 and 3 2).
  - **Increasing strip**: strip of elements in increasing order (e.g. 7 8)

```
0 1 9 4 3 7 8 2 5 6 10
```

- A *single-element strip* can be declared either increasing or decreasing. We will choose to declare them as **decreasing** with exception of extension strips (with 0 and \(n+1\))
If permutation $\pi$ contains at least one decreasing strip, then there exists a reversal $\rho$ which decreases the number of breakpoints (i.e. $b(\pi \cdot \rho) < b(\pi)$).

Consider $\pi = 1 \ 4 \ 6 \ 5 \ 7 \ 8 \ 3 \ 2 \ 0$  

$\quad 0 \mid 1 \mid 4 \mid 6 \mid 5 \mid 7 \mid 8 \mid 3 \mid 2 \mid 9 \quad b(\pi) = 5$
Things to Consider

Consider \( \pi = 1 \ 4 \ 6 \ 5 \ 7 \ 8 \ 3 \ 2 \)

\[
\begin{array}{cccccccc}
0 & 1 & 4 & 6 & 5 & 7 & 8 & 3 & 2 & 9 \\
\end{array}
\]

\( b(\pi) = 5 \)

- Choose the decreasing strip with the smallest element \( k \) in \( \pi \) \((it’ll always be the rightmost)\)
- Find \( k-1 \) in the permutation \\
  \((it’ll always be flanked by a breakpoint)\)
- Reverse the segment between \( k \) and \( k-1 \)

Thus, removing the breakpoint flanking \( k-1 \)
Things to Consider

Consider \( \pi = 1 4 6 5 7 8 3 2 \)

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

- Choose the decreasing strip with the smallest element \( k \) in \( \pi \) (it’ll always be the rightmost)
- Find \( k - 1 \) in the permutation
  (it’ll always be flanked by a breakpoint)
- Reverse the segment between \( k \) and \( k-1 \)
- Repeat until there is no decreasing strip

reduced by 1!
Things to Consider

Consider \( \pi = 1 \ 4 \ 6 \ 5 \ 7 \ 8 \ 3 \ 2 \)

\[ 0 \ 1 \ 2 \ 3 | 8 \ 7 \ 5 \ 6 | 4 | 9 \]

\( b(\pi) = 4 \)

- Choose the decreasing strip with the smallest element \( k \) in \( \pi \) (it’ll always be the rightmost)
- Find \( k - 1 \) in the permutation (it’ll always be flanked by a breakpoint)
- Reverse the segment between \( k \) and \( k-1 \)
- Repeat until there is no decreasing strip
Things to Consider

Consider \( \pi = 1 \ 4 \ 6 \ 5 \ 7 \ 8 \ 3 \ 2 \)

- Choose the decreasing strip with the smallest element \( k \) in \( \pi \) (it’ll always be the rightmost)
- Find \( k - 1 \) in the permutation (it’ll always be flanked by a breakpoint)
- Reverse the segment between \( k \) and \( k-1 \)
- Repeat until there is no decreasing strip
Things to Consider

Consider $\pi = 1 4 6 5 7 8 3 2$

- Choose the decreasing strip with the smallest element $k$ in $\pi$ (it’ll always be the rightmost)
- Find $k - 1$ in the permutation (it’ll always be flanked by a breakpoint)
- Reverse the segment between $k$ and $k-1$
- Repeat until there is no decreasing strip

$b(\pi) = 2$
Things to Consider

Consider \( \pi = 1 \, 4 \, 6 \, 5 \, 7 \, 8 \, 3 \, 2 \)

No breakpoint left!

\[
\begin{array}{cccccccc}
0 & 1 & 2 & 3 & 4 & 5 & 6 & 7 & 8 & 9 \\
\end{array}
\]

- Choose the decreasing strip with the smallest element \( k \) in \( \pi \) (it’ll always be the rightmost)
- Find \( k - 1 \) in the permutation (it’ll always be flanked by a breakpoint)
- Reverse the segment between \( k \) and \( k-1 \)
- Repeat until there is no decreasing strip
Consider $\pi = 1 4 6 5 7 8 3 2$

$b(\pi) = 5$

$b(\pi) = 4$

$b(\pi) = 2$

$b(\pi) = 0$

d(\pi) = 3

Does it work for any permutation?
• If there is no decreasing strip, there may be no strip-reversal $\rho$ that reduces the number of breakpoints (i.e. $b(\pi \cdot \rho) \geq b(\pi)$ for any reversal $\rho$).

• However, reversing an increasing strip creates a decreasing strip, and the number of breakpoints remains unchanged.

• Then the number of breakpoints will be reduced in the following steps.
• If there is no decreasing strip, there may be no strip-reversal $\rho$ that reduces the number of breakpoints (i.e. $b(\pi \cdot \rho) \geq b(\pi)$ for any reversal $\rho$).

• However, reversing an increasing strip creates a decreasing strip, and the number of breakpoints remains unchanged.

• Then the number of breakpoints will be reduced in the following steps.
Potential Gotcha

- If there is no decreasing strip, there may be no strip-reversal $\rho$ that reduces the number of breakpoints (i.e. $b(\pi \cdot \rho) \geq b(\pi)$ for any reversal $\rho$).
- However, reversing an increasing strip creates a decreasing strip, and the number of breakpoints remains unchanged.
- Then the number of breakpoints will be reduced in the following steps.
If there is no decreasing strip, there may be no strip-reversal \( \rho \) that reduces the number of breakpoints (i.e. \( b(\pi \cdot \rho) \geq b(\pi) \) for any reversal \( \rho \)).

However, reversing an increasing strip creates a decreasing strip, and the number of breakpoints remains unchanged.

Then the number of breakpoints will be reduced in the following steps.
If there is no decreasing strip, there may be no strip-reversal $\rho$ that reduces the number of breakpoints (i.e. $b(\pi \cdot \rho) \geq b(\pi)$ for any reversal $\rho$).

However, reversing an increasing strip creates a decreasing strip, and the number of breakpoints remains unchanged.

Then the number of breakpoints will be reduced in the following steps.
ImprovedBreakpointReversalSort

**ImprovedBreakpointReversalSort**($\pi$)

1. **while** $b(\pi) > 0$
2. **if** $\pi$ has a decreasing strip
3. Among all possible reversals, choose reversal $\rho$
   that **minimizes** $b(\pi \cdot \rho)$
4. **else**
5. Choose a reversal $\rho$ that flips an increasing strip in $\pi$
6. $\pi \leftarrow \pi \cdot \rho$
7. **output** $\pi$
8. **return**
```python
def improvedBreakpointReversalSort(seq):
    while hasBreakpoints(seq):
        increasing, decreasing = getStrips(seq)
        if len(decreasing) > 0:
            reversal = pickReversal(seq, decreasing)
        else:
            reversal = increasing[0]
        print seq, "reversal", reversal
        seq = doReversal(seq, reversal)
        print seq, "Sorted"
    return
```
Performance

- ImprovedBreakPointReversalSort is an approximation algorithm with a performance guarantee of no worse than 4
  - It eliminates at least one breakpoint in every two steps; at most $2b(\pi)$ steps
  - Optimal algorithm eliminates at most 2 breakpoints in every step: $d(\pi) \geq b(\pi) / 2$
  - Approximation ratio:
    \[
    \frac{2b(\pi)}{d(\pi)} \leq \frac{2b(\pi)}{b(\pi)} \leq 4
    \]
A Better Approximation Ratio

• If there is a decreasing strip, the next reversal reduces $b(\pi)$ by at least one.

• The only bad case is when there is no decreasing strip, as then we need a reversal that does not reduce $b(\pi)$.
  – If we could always choose a reversal reducing $b(\pi)$ and, at the same time, yielding a permutation that again has at least one decreasing strip, the bad case would never occur.
  – If all reversals that reduce $b(\pi)$ create a permutation without decreasing strips, then there exists a reversal that reduces $b(\pi)$ by two?!
  – When the algorithm creates a permutation without decreasing strip, the previous reversal must have reduced $b(\pi)$ by two.

• At most $b(\pi)$ reversals are needed.

• Approximation ratio: $\frac{b(\pi)}{d(\pi)} \leq \frac{b(\pi)}{2} = 2$
Both are Greedy Algorithms

- **SimpleReversalSort**
  - Attempts to maximize \( \text{prefix}(\pi) \) at each step
  - Performance guarantee: \( \frac{n-1}{2} \)

- **ImprovedBreakPointReversalSort**
  - Attempts to reduce the number of breakpoints at each step
  - Performance guarantee: 2
Try it yourself

0 1|3|8 7 6|2|4 5|9 10
Next Time

- Dynamic Programming Algorithms