Algorithms and Data Structures in Biology

Greedy Algorithms

Ugo Dal Lago

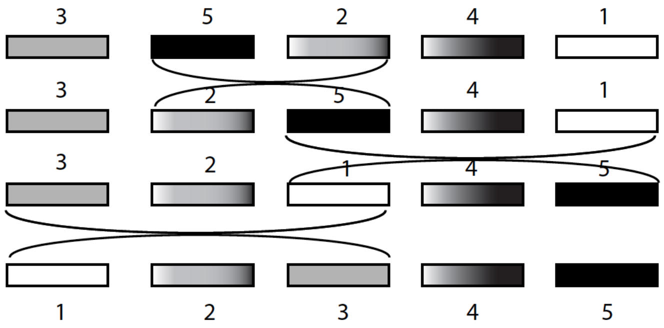
University of Bologna, Academic Year 2018/2019

The Greedy Paradigm

- Greedy algorithms proceed by making choices which are **locally optimal**.
  - As such, greedy algorithms are *not* guaranteed to produce the correct output.
- This way, however, greedy algorithms keep their complexity under control.
  - Very often, greedy algorithms for hard combinatorial problems are known having **polynomial time** complexity.
- A typical example of a greedy algorithm is BetterChange which, as know, does not necessarily produce an optimal solution.

Genome Rearrangements

Mouse X Chromosome



Human X Chromosome

Modeling Genome Rearrangements

- The order of syntheny blocks in a piece of genome can be represented by a permutation $\sigma : \{1, \ldots, n\} \to \{1, \ldots, n\}$, itself representable as a sequence

$$\pi = \pi_1 \; \pi_2 \; \cdots \; \pi_n \tag{1}$$

  where $\pi_1, \ldots, \pi_n \in \{1, \ldots, n\}$ are all distinct.

- A *reversal* $\rho(i, j)$ (where $1 \leq i < j \leq n$) has the effect of reversing the order of

$$\pi_i \; \pi_{i+1} \; \cdots \; \pi_j$$

  trasforming π as in (1) into

$$\pi_1 \; \cdots \; \pi_{i-1} \; (\pi_j \; \pi_{j-1} \cdots \pi_{i+1} \; \pi_i) \; \pi_{j+1} \cdots \pi_n$$

  which we indicate as $\pi \cdot \rho$.

Modeling Genome Rearrangements

- We could start from

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

and apply to it the reversal $\rho(3,6)$ obtaining the sequence

$$\pi \cdot \rho(3,6) = 1\ 2\ (5\ 7\ 3\ 4)\ 6$$

- Biologists are often interested in the most *parsimonious* evolutionary scenario, in which a chromosome evolves into another one by **very few rearrangements**, i.e., by very few reversals.

Modeling Genome Rearrangements

- We could start from

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

  and apply to it the reversal $\rho(3,6)$ obtaining the sequence

$$\pi \cdot \rho(3,6) = 1\ 2\ (5\ 7\ 3\ 4)\ 6$$

- Biologists are often interested in the most *parsimonious* evolutionary scenario, in which a chromosome evolves into another one by **very few rearrangements**, i.e., by very few reversals.

Modeling Genome Rearrangements

- We could start from

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

  and apply to it the reversal $\rho(3,6)$ obtaining the sequence

$$\pi \cdot \rho(3,6) = 1\ 2\ (5\ 7\ 3\ 4)\ 6$$

- Biologists are often interested in the most *parsimonious* evolutionary scenario, in which a chromosome evolves into another one by **very few rearrangements**, i.e., by very few reversals.

The Reversal Distance and Sorting by Distance Problems

Reversal Distance Problem:
Given two permutations, find a shortest series of reversals that transforms one permutation into another.

Input: Permutations π and σ.

Output: A series of reversals $\rho_1, \rho_2, \ldots, \rho_t$ transforming π into σ (i.e., $\pi \cdot \rho_1 \cdot \rho_2 \cdots \rho_t = \sigma$), such that t is minimum.

The Reversal Distance and Sorting by Distance Problems

Reversal Distance Problem:
Given two permutations, find a shortest series of reversals that transforms one permutation into another.

Input: Permutations π and σ.

Output: A series of reversals $\rho_1, \rho_2, \ldots, \rho_t$ transforming π into σ (i.e., $\pi \cdot \rho_1 \cdot \rho_2 \cdots \rho_t = \sigma$), such that t is minimum.

Sorting by Reversals Problem:
Given a permutation, find a shortest series of reversals that transforms it into the identity permutation.

Input: Permutation π.

Output: A series of reversals $\rho_1, \rho_2, \ldots, \rho_t$ transforming π into the identity permutation such that t is minimum.

Sorting by Distance

- The latter problem is a *special case* of the former, so starting with it is a good idea.
- As the name implies, the sorting by distance problem is a special case of the sorting problem, which is also an optimization problem.
- What if we just produce in output a sequence of at most n reversals, each of them bringing i to i-th position?

Sorting by Distance

- The latter problem is a *special case* of the former, so starting with it is a good idea.
- As the name implies, the sorting by distance problem is a special case of the sorting problem, which is also an optimization problem.
- What if we just produce in output a sequence of at most n reversals, each of them bringing i to i-th position?

Sorting by Distance

- The latter problem is a *special case* of the former, so starting with it is a good idea.
- As the name implies, the sorting by distance problem is a special case of the sorting problem, which is also an optimization problem.
- What if we just produce in output a sequence of at most n reversals, each of them bringing i to i-th position?

Sorting by Distance

- The latter problem is a *special case* of the former, so starting with it is a good idea.
- As the name implies, the sorting by distance problem is a special case of the sorting problem, which is also an optimization problem.
- What if we just produce in output a sequence of at most n reversals, each of them bringing i to i-th position?

```
SimpleReversalSort($\pi$)
for $i \leftarrow 1$ to $n - 1$
    $j \leftarrow$ position of element $i$ in $\pi$ (i.e., $\pi_j = i$)
    if $j \neq i$
        $\pi \leftarrow \pi \cdot \rho(i, j)$
        output $\pi$
    if $\pi$ is the identity permutation
        return
```

Reversal Sort is not Always Correct

- It is easy to realize that
  - the reversals produced in output by SimpleReversalSort trasform the input into the identity permutation
  - the number of such reversals is not always the minimum
- Consider, as an example

$$\underline{6\,1}\,2\,3\,4\,5 \rightarrow 1\,\underline{6\,2}\,3\,4\,5 \rightarrow 1\,2\,\underline{6\,3}\,4\,5 \rightarrow 1\,2\,3\,\underline{6\,4}\,5 \rightarrow 1\,2\,3\,4\,\underline{6\,5} \rightarrow 1\,2\,3\,4\,5\,6$$

and compare it to

$$\underline{6\,1\,2\,3\,4\,5} \rightarrow \underline{5\,4\,3\,2\,1}\,6 \rightarrow 1\,2\,3\,4\,5\,6.$$

- More generally SimpleReversalSort when applied to $n\;1\;2\;\cdots(n-1)$ produces $n-1$ reversals, while such a permutation can be ordered in just two steps.

Reversal Sort is not Always Correct

- It is easy to realize that
  - the reversals produced in output by SimpleReversalSort trasform the input into the identity permutation
  - the number of such reversals is not always the minimum
- Consider, as an example

$$\underline{6\,1}\,2\,3\,4\,5 \rightarrow 1\,\underline{6\,2}\,3\,4\,5 \rightarrow 1\,2\,\underline{6\,3}\,4\,5 \rightarrow 1\,2\,3\,\underline{6\,4}\,5 \rightarrow 1\,2\,3\,4\,\underline{6\,5} \rightarrow 1\,2\,3\,4\,5\,6$$

  and compare it to

$$\underline{6\,1\,2\,3\,4\,5} \rightarrow \underline{5\,4\,3\,2\,1}\,6 \rightarrow 1\,2\,3\,4\,5\,6.$$

- More generally SimpleReversalSort when applied to $n\ 1\ 2\ \cdots (n-1)$ produces $n-1$ reversals, while such a permutation can be ordered in just two steps.

Reversal Sort is not Always Correct

- It is easy to realize that
  - the reversals produced in output by SimpleReversalSort trasform the input into the identity permutation
  - the number of such reversals is not always the minimum
- Consider, as an example

$$\underline{6\,1}\,2\,3\,4\,5 \rightarrow 1\,\underline{6\,2}\,3\,4\,5 \rightarrow 1\,2\,\underline{6\,3}\,4\,5 \rightarrow 1\,2\,3\,\underline{6\,4}\,5 \rightarrow 1\,2\,3\,4\,\underline{6\,5} \rightarrow 1\,2\,3\,4\,5\,6$$

  and compare it to

$$\underline{6\,1\,2\,3\,4\,5} \rightarrow \underline{5\,4\,3\,2\,1}\,6 \rightarrow 1\,2\,3\,4\,5\,6.$$

- More generally SimpleReversalSort when applied to $n\;1\;2\;\cdots(n-1)$ produces $n-1$ reversals, while such a permutation can be ordered in just two steps.

Approximation Algorithms

- SimpleReversalSort is a greedy algorithm, because it "solves" the underlying combinatorial problem by making some choices which are *locally* good, although being *globally* bad.
- It is also an **approximation algorithm**, namely an algorithm that gives an approximate solution to an optimization problem:
  - Although the output is *correct*, it does not have the *minimum* length.
- How could we evaluate the **quality** of an approximation algorithm?
  - We would like it to output solutions which, although not optimal, are not **too far** from being optimal.
  - But how could we measure the distance between any solution and the optimal one?

Approximation Algorithms

- SimpleReversalSort is a greedy algorithm, because it "solves" the underlying combinatorial problem by making some choices which are *locally* good, although being *globally* bad.
- It is also an **approximation algorithm**, namely an algorithm that gives an approximate solution to an optimization problem:
  - Although the output is *correct*, it does not have the *minimum* length.
- How could we evaluate the **quality** of an approximation algorithm?
  - We would like it to output solutions which, although not optimal, are not **too far** from being optimal.
  - But how could we measure the distance between any solution and the optimal one?

Approximation Ratios

- Given an approximation algorithm $\mathcal{A}$ and a problem instance π, we define:
  - The *optimal* value $OPT(\pi)$ as the optimal value for the problem instance π
  - The *approximation ratio of* $\mathcal{A}$ *on* π as
$$AR(\pi) = \frac{\mathcal{A}(\pi)}{OPT(\pi)}.$$
  - The *approximation ratio of* $\mathcal{A}$ as $\mathcal{A}(\pi)/OPT(\pi)$ as the function associating
$$\max_{|\pi|=n} AR(\pi) \text{ or } \min_{|\pi|=n} AR(\pi)$$
    to n (depending on the nature of the optimization problem).
- As an example, if $\mathcal{A}$ is SimpleReversalSort, then
$$\max_{|\pi|=n} AR(\pi) = \max_{|\pi|=n} \frac{\mathcal{A}(\pi)}{OPT(\pi)} \geq \frac{n-1}{2}$$

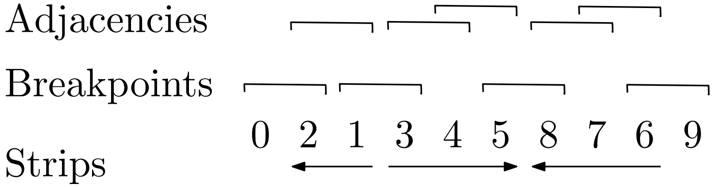
Adjacencies, Breakpoints, and Strips

- In the following, it is convenient to reprsent a permutation on $\{1, \ldots, n\}$ as a sequence $\pi = \pi_0 \, \pi_1 \, \cdot \, \pi_n \pi_{n+1}$, where $\pi_0 = 0$ and $\pi_{n+1} = n + 1$.
- Given such a π, a pair of neighboring elements π_i, π_{i+1} (for $0 \leq i \leq n$) is said to be:
  - An **adjacency** if π_i, π_{i+1} are consecutive numbers;
  - A **breakpoint** otherwise.
- The sequence π can have any number of breakpoints, indicated as $b(\pi)$, included between 0 and n.
  - In the identity permutation $b(\pi) = 0$.
  - Any reverse can make $b(\pi)$ to decrease by at most 2 (and, indeed, $d(\pi) \geq \frac{b(\pi)}{2}$.
- A **strip** in π is any interval between two consecutive breakpoints, i.e., any maximal segment of π without breakpoints.

Adjacencies, Breakpoints, and Strips

- In the following, it is convenient to reprsent a permutation on $\{1, \ldots, n\}$ as a sequence $\pi = \pi_0\ \pi_1\ \cdot\ \pi_n \pi_{n+1}$, where $\pi_0 = 0$ and $\pi_{n+1} = n + 1$.
- Given such a π, a pair of neighboring elements π_i, π_{i+1} (for $0 \leq i \leq n$) is said to be:
  - An **adjacency** if π_i, π_{i+1} are consecutive numbers;
  - A **breakpoint** otherwise.
- The sequence π can have any number of breakpoints, indicated as $b(\pi)$, included between 0 and n.
  - In the identity permutation $b(\pi) = 0$.
  - Any reverse can make $b(\pi)$ to decrease by at most 2 (and, indeed, $d(\pi) \geq \frac{b(\pi)}{2}$.
- A **strip** in π is any interval between two consecutive breakpoints, i.e., any maximal segment of π without breakpoints.

Adjacencies, Breakpoints, and Strips



Forcing $b(\pi)$ to Decrease

- We can take $b(\pi)$ as a measure of *how far* we are from the identity.
- This suggests the following algorithm:

```
BREAKPOINTREVERSALSORT($\pi$)
1  while $b(\pi) > 0$
2        Among all reversals, choose reversal $\rho$ minimizing $b(\pi \cdot \rho)$
3        $\pi \leftarrow \pi \cdot \rho$
4        output $\pi$
5  return
```

- There are several problems with BREAKPOINTREVERSALSORT:
  - Why does it *terminate*?
  - Can we give an *(over)estimate* to the number of iterations?

Forcing $b(\pi)$ to Decrease

- We can take $b(\pi)$ as a measure of *how far* we are from the identity.
- This suggests the following algorithm:

```
BreakpointReversalSort($\pi$)
1  while $b(\pi) > 0$
2      Among all reversals, choose reversal $\rho$ minimizing $b(\pi \cdot \rho)$
3      $\pi \leftarrow \pi \cdot \rho$
4      output $\pi$
5  return
```

- There are several problems with BreakpointReversalSort:
  - Why does it *terminate*?
  - Can we give an *(over)estimate* to the number of iterations?

Forcing $b(\pi)$ to Decrease

- Answers to the questions above can be given by analysing strips and in particular *decreasing* strips, rather than breakpoints.

Theorem

If a permutation π contains a decreasing strip, then there is a reversal ρ that decreases the number of breakpoints in π, that is $b(\pi \cdot \rho) < b(\pi)$.

$$\begin{array}{ll}
(\underrightarrow{0}\ \underleftarrow{8}\ \underleftarrow{2}\ \underleftarrow{7\ 6\ 5}\ \underleftarrow{1}\ \underleftarrow{4}\ \underleftarrow{3}\ \underrightarrow{9}) & b(\pi) = 6 \\
(\underrightarrow{0}\ \underleftarrow{2}\ \underleftarrow{8\ 7\ 6\ 5}\ \underleftarrow{1}\ \underleftarrow{4}\ \underleftarrow{3}\ \underrightarrow{9}) & b(\pi) = 5 \\
(\underrightarrow{0}\ \underrightarrow{2\ 3\ 4}\ \underleftarrow{1}\ \underrightarrow{5\ 6\ 7\ 8\ 9}) & b(\pi) = 3 \\
(\underrightarrow{0}\ \underleftarrow{4\ 3\ 2\ 1}\ \underrightarrow{5\ 6\ 7\ 8\ 9}) & b(\pi) = 2 \\
(\underrightarrow{0\ 1\ 2\ 3\ 4\ 5\ 6\ 7\ 8\ 9}) & b(\pi) = 0
\end{array}$$

Forcing $b(\pi)$ to Decrease

- Answers to the questions above can be given by analysing strips and in particular *decreasing* strips, rather than breakpoints.

Theorem

If a permutation π contains a decreasing strip, then there is a reversal ρ that decreases the number of breakpoints in π, that is $b(\pi \cdot \rho) < b(\pi)$.

$$\begin{array}{ll}
(\underrightarrow{0}\ \underleftarrow{8}\ \underleftarrow{2}\ \underleftarrow{7\ 6\ 5}\ \underleftarrow{1}\ \underleftarrow{4}\ \underleftarrow{3}\ \underrightarrow{9}) & b(\pi) = 6 \\
(\underrightarrow{0}\ \underleftarrow{2}\ \underleftarrow{8\ 7\ 6\ 5}\ \underleftarrow{1}\ \underleftarrow{4}\ \underleftarrow{3}\ \underrightarrow{9}) & b(\pi) = 5 \\
(\underrightarrow{0}\ \ \underrightarrow{2\ \ 3\ \ 4}\ \ \underleftarrow{1}\ \ \underrightarrow{5\ \ 6\ \ 7\ \ 8\ \ 9}) & b(\pi) = 3 \\
(\underrightarrow{0}\ \ \underleftarrow{4\ \ 3\ \ 2\ \ 1}\ \underrightarrow{5\ \ 6\ \ 7\ \ 8\ \ 9}) & b(\pi) = 2 \\
(\underrightarrow{0\ \ 1\ \ 2\ \ 3\ \ 4\ \ 5\ \ 6\ \ 7\ \ 8\ \ 9}) & b(\pi) = 0
\end{array}$$

Forcing $b(\pi)$ to Decrease

- Answers to the questions above can be given by analysing strips and in particular *decreasing* strips, rather than breakpoints.

Theorem

If a permutation π contains a decreasing strip, then there is a reversal ρ that decreases the number of breakpoints in π, that is $b(\pi \cdot \rho) < b(\pi)$.

$$
\begin{array}{ll}
(\underrightarrow{0}\ \underleftarrow{8}\ \underleftarrow{2}\ \underleftarrow{7\,6\,5}\ \underleftarrow{1}\ \underleftarrow{4}\ \underleftarrow{3}\ \underrightarrow{9}) & b(\pi) = 6 \\
(\underrightarrow{0}\ \underleftarrow{2}\ \underleftarrow{8\,7\,6\,5}\ \underleftarrow{1}\ \underleftarrow{4}\ \underleftarrow{3}\ \underrightarrow{9}) & b(\pi) = 5 \\
(\underrightarrow{0}\ \underrightarrow{2\ 3\ 4}\ \underleftarrow{1}\ \underrightarrow{5\ 6\ 7\ 8\ 9}) & b(\pi) = 3 \\
(\underrightarrow{0}\ \underleftarrow{4\ 3\ 2\ 1}\ \underrightarrow{5\ 6\ 7\ 8\ 9}) & b(\pi) = 2 \\
(\underrightarrow{0\ 1\ 2\ 3\ 4\ 5\ 6\ 7\ 8\ 9}) & b(\pi) = 0
\end{array}
$$

An Improved Greedy Algoritm

```
ImprovedBreakpointReversalSort($\pi$)
1 while $b(\pi) > 0$
2     if $\pi$ has a decreasing strip
3         Among all reversals, choose reversal $\rho$ minimizing $b(\pi \cdot \rho)$
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
```

- Could we get an upper bound on the approximation ratio for this algorithm?

Theorem

The algorithm ImprovedBreakpointReversalSort *has an approximation ratio of at most 4.*

An Improved Greedy Algoritm

```
IMPROVEDBREAKPOINTREVERSALSORT($\pi$)
1  while $b(\pi) > 0$
2      if $\pi$ has a decreasing strip
3          Among all reversals, choose reversal $\rho$ minimizing $b(\pi \cdot \rho)$
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
```

- Could we get an upper bound on the approximation ratio for this algorithm?

Theorem

The algorithm IMPROVEDBREAKPOINTREVERSALSORT *has an approximation ratio of at most* 4*.*

What About Motif Finding?

- A problem for which we have only given exhaustive search algorithms is *motif finding*.
  - We have also given branch-and-bound techniques for it, but as we know, the complexity stays essentially the same.
- Could the *greedy approach* be applied to motif finding? What can be greedy about the underlying combinatorial problem?
  - We could choose which positions are "the good ones" for *the first two strings*, without looking at the other ones.
  - Once a reference string has been choosen, the other strings (from the third to the last) are considered one after another, looking for kthe best position which maximizes the partial score.
- The obtained algorithm works in polynomial time, but no bound is known on its approximation ratio.
  - The algorithm is however very useful in practice, being a good compromise between *performance* and *accuracy*.

What About Motif Finding?

- A problem for which we have only given exhaustive search algorithms is *motif finding*.
  - We have also given branch-and-bound techniques for it, but as we know, the complexity stays essentially the same.
- Could the *greedy approach* be applied to motif finding? What can be greedy about the underlying combinatorial problem?
  - We could choose which positions are "the good ones" for *the first two strings*, without looking at the other ones.
  - Once a reference string has been choosen, the other strings (from the third to the last) are considered one after another, looking for kthe best position which maximizes the partial score.
- The obtained algorithm works in polynomial time, but no bound is known on its approximation ratio.
  - The algorithm is however very useful in practice, being a good compromise between *performance* and *accuracy*.

What About Motif Finding?

- A problem for which we have only given exhaustive search algorithms is *motif finding*.
  - We have also given branch-and-bound techniques for it, but as we know, the complexity stays essentially the same.
- Could the *greedy approach* be applied to motif finding? What can be greedy about the underlying combinatorial problem?
  - We could choose which positions are “the good ones” for *the first two strings*, without looking at the other ones.
  - Once a reference string has been choosen, the other strings (from the third to the last) are considered one after another, looking for kthe best position which maximizes the partial score.
- The obtained algorithm works in polynomial time, but no bound is known on its approximation ratio.
  - The algorithm is however very useful in practice, being a good compromise between *performance* and *accuracy*.

An Improved (but Greedy) Algorithm for Motif Finding

```
GreedyMotifSearch($DNA, t, n, l$)
bestMotif $\leftarrow (1, 1, \ldots, 1)$
s $\leftarrow (1, 1, \ldots, 1)$
for $s_1 \leftarrow 1$ to $n - l + 1$
    for $s_2 \leftarrow 1$ to $n - l + 1$
        if $Score(\mathbf{s}, 2, DNA) > Score(\mathbf{bestMotif}, 2, DNA)$
            $BestMotif_1 \leftarrow s_1$
            $BestMotif_2 \leftarrow s_2$
$s_1 \leftarrow BestMotif_1$
$s_2 \leftarrow BestMotif_2$
for $i \leftarrow 3$ to $t$
    for $s_i \leftarrow 1$ to $n - l + 1$
        if $Score(\mathbf{s}, i, DNA) > Score(\mathbf{bestMotif}, i, DNA)$
            $bestMotif_i \leftarrow s_i$
    $s_i \leftarrow bestMotif_i$
return bestMotif
```

Thank You!

Questions?