Algorithms and bioinformatics
Comparative genomics

Anthony Labarre

September 12, 2016
Context and motivations

- **Deoxyribonucleic acid**: double helix of *nucleotides* (A, C, G, T);
- Complementarity (A-T, C-G): one strip is enough;
- *Gene* = sequence of nucleotides (that codes for a specific protein);
- *Chromosome* = ordered set of genes;
- *Genome* = set of chromosomes;
- **Goal**: compare genomes;
Context and motivations

- Biologists are interested in comparing species, for example:
  - in order to classify them;
  - in order to explain evolution by reconstructing scenarios;
- (Dis)similarity measures are needed;
- Usually based on the sequenced genomes;
At the nucleotide level

- Most comparisons take place at the nucleotide level;

Example (sequence alignment)

\[ S_1 : \cdots \ T \ C \ C \ G \ C \ C \ C \ A \ - \ - \ C \ T \ A \ \cdots \]
\[ S_2 : \cdots \ T \ C \ G \ G \ A \ C \ T \ G \ G \ C \ - \ A \ \cdots \]

- Matches, substitutions, insertions and deletions;
- Correspond to mutations;
At the “gene” level

- Some mutations act on *segments* of nucleotides;
- Those large-scale mutations are called **genome rearrangements**;
- Sequence alignment becomes unfit;

**Example (genomes as sequences of segments)**

![Diagram showing genome rearrangements](image)
Genome rearrangements

▶ Our problem:

**Problem (PAIRWISE GENOME REARRANGEMENT)**

**Input:** genomes $G_1$, $G_2$, a set $S$ of mutations;

**Goal:** find a shortest sequence of elements of $S$ that transforms $G_1$ into $G_2$.

▶ Related, simpler problem: compute the **evolutionary distance** $d_S(G_1, G_2)$ (i.e. just the **length** of a shortest sequence);

▶ Many variants, depending on how genomes are modelled, what (and how) mutations are taken into account, etc.;
Modelling genomes as permutations

- Genomes are seen as permutations if:
  1. they form ordered sequences of genes (or other segments), and
  2. they only differ by order (no duplications or deletions).

Example (genomes → permutations)

(A) 5 1 2 4 7 3 6
     \         \       \     \        \       \\
    1 2 3 4 5 6 7 (B)
Genome rearrangements for permutations

- Segments can be numbered as we wish, so we assume either genome is the identity permutation \( \iota = \langle 1 \ 2 \ \cdots \ n \rangle \) and we wish to sort the other genome:

**Problem (GENOME REARRANGEMENT (PERMUTATIONS))**

**Input:** a permutation \( \pi \) in \( S_n \), a set \( S \subseteq S_n \) of (per)mutations;

**Goal:** find a shortest sorting sequence of elements of \( S \) for \( \pi \).

- Again, we can also focus on merely computing \( d_S(\pi) \) – the length of an optimal sorting sequence;

- \( S \) must generate \( S_n \) for any pair of permutations to be a finite distance apart;
Notation and definitions pertaining to permutations

- Permutations can be written in one- or two-row notation:

\[ \pi = \begin{pmatrix}
1 & 2 & 3 & 4 & 5 & 6 \\
4 & 1 & 6 & 2 & 5 & 3
\end{pmatrix} = \langle 4 \ 1 \ 6 \ 2 \ 5 \ 3 \rangle. \]

- We deal exclusively with \([n] = \{1, 2, \ldots, n\}\);

- All permutations of \([n]\) with composition form the symmetric group \(S_n\);

- Composition: the usual \(\circ\), which means that in \(\pi \circ \sigma\), \(\sigma\) is applied first;
Sorting permutations by adjacent exchanges

- Simple operation: exchange any two adjacent elements:

Example

\[
\begin{array}{ccccccc}
4 & 1 & 6 & 2 & 5 & 3 \\
1 & 4 & 6 & 2 & 5 & 3 \\
\end{array}
\]

- So we want to sort a permutation by performing as few such exchanges as possible;

- Solution:
  - sorting: use Bubble Sort;
  - distance: the number of swaps used by Bubble Sort;
A closed formula for the adjacent exchange distance

- An inversion in a permutation is a pair of misplaced elements: \((\pi_i, \pi_j)\) with \(i < j\) and \(\pi_i > \pi_j\);

- The permutation graph of \(\pi\) has \([n]\) as vertex set and the inversions of \(\pi\) as edges;

Example (the permutation graph of \(\langle 4 \ 1 \ 6 \ 2 \ 5 \ 3 \rangle\))

```
        6
       / \  \
      4   5
     / \  / \
    2   1 3
   /   /   /
  3   5   6
```
Theorem

The adjacent exchange distance of $\pi$ is $|\text{Inv}(\pi)| = |E(PG(\pi))|$.  

Proof sketch.

Each adjacent swap “fixes” at most one inversion, which is equivalent to removing an edge from $PG$, and we can always find such a move at every step if $\pi \neq \iota$.  

Sorting permutations by exchanges

- Simple operation: exchange any two elements:

Example

```
4  1  6  2  5  3
2  1  6  4  5  3
```

- So we want to sort a permutation by performing as few such exchanges as possible;
Sorting permutations by exchanges

Here’s a more complete example:

Example (a sorting sequence for \langle 4 \ 1 \ 6 \ 2 \ 5 \ 3 \rangle)

\[
\begin{array}{cccccc}
4 & 1 & 6 & 2 & 5 & 3 \\
1 & 4 & 6 & 2 & 5 & 3 \\
1 & 2 & 6 & 4 & 5 & 3 \\
1 & 2 & 3 & 4 & 5 & 6 \\
\end{array}
\]

It works... but can we do better?
Sorting permutations by exchanges

- Our goal: each element should be “at the right place”;
- Some elements are already where they should be, so they won’t move;
- Strategy: read permutation from left to right, and:
  - if $\pi_i = i$, pass;
  - otherwise, exchange $\pi_i$ with $i$;
Sorting permutations by exchanges

- The algorithm obviously terminates;
- At every step, we “fix” one or two positions;
- We use the minimum number of exchanges;
- On the other hand, we’d like to be able to compute the distance without sorting;
Cycles

- Computing the distance requires using the **cycles** of the permutation;
- Those cycles are obtained by iterating the permutation’s action on \{1, 2, \ldots, n\}, stopping when all elements have been visited;

**Example (cycles of \langle 4 \ 1 \ 6 \ 2 \ 5 \ 3 \rangle)**

```
\[
\begin{array}{cccccc}
1 & & 2 & & 3 & 4 \\
\downarrow & \leftarrow & \downarrow & \rightarrow & \downarrow & \downarrow \\
4 & & 1 & & 6 & 2 \\
\downarrow & \downarrow & \downarrow & \rightarrow & \downarrow & \downarrow \\
5 & & 5 & & 3 &
\end{array}
\]```

Disjoint cycle decomposition of permutations

- Each permutation decomposes into disjoint cycles:

\[
\pi = \langle 1 \ 2 \ 3 \ 4 \ 5 \ 6 \rangle = (1, 4, 2)(3, 6)(5).
\]

- The graph of the permutation \( \pi \), denoted by \( \Gamma(\pi) \), pictures this decomposition:

- The number of cycles of \( \pi \) is written \( c(\pi) \);
- 1-cycles are sometimes omitted;
Cycles and sorting

- Cycles of length 1 correspond to sorted elements; all other cycles consist of elements that are misplaced;

\[
(4 \rightarrow 1 \rightarrow 6 \rightarrow 2 \rightarrow 5 \rightarrow 3) \rightarrow (1 \rightarrow 2 \rightarrow 3 \rightarrow 4 \rightarrow 5 \rightarrow 6)
\]

- Sorting comes down to splitting cycles until we only have cycles of length 1;
- Our algorithm repeatedly splits \(k\)-cycles into a 1-cycle and a \((k - 1)\)-cycle;
Computing the “exchange distance” \( \text{exc}(\cdot) \)

- At each step, we can always create a new cycle if \( \pi \neq \iota \), so:

\[
\text{exc}(\pi) \leq n - c(\pi)
\]

- And we can’t do better, so:

\[
\text{exc}(\pi) \geq n - c(\pi)
\]

- Therefore:

**Theorem ([Cayley, 1849])**

*The exchange distance of \( \pi \) in \( S_n \) is \( n - c(\pi) \).*
Lessons from sorting by exchanges

- Note that $\iota$ is the only permutation with $n$ cycles;
- The formula $\text{exc}(\pi) = n - c(\pi)$ expresses:
  - the difference between the number of cycles we have and the number of cycles we want;
  - and the fact that at each step, we can obtain exactly one new cycle.
- This point of view will be crucial to sorting problems;
The mutations we observe in evolution (may) act on intervals;

We’ll now look at two generalisations of exchanges:

1. block-interchanges;
2. transpositions;

**Block-interchanges** exchange two disjoint intervals in a permutation;

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

**Transpositions** displace an interval of the permutation;

\[ \begin{array}{cccccccc}
1 & 2 & 3 & 4 & 5 & 6 & 7 & 8 & 9 & 10 \\
\end{array} \rightarrow \begin{array}{cccccccc}
1 & 5 & 6 & 7 & 8 & 2 & 3 & 4 & 9 & 10 \\
\end{array} \]
Computing the associated distances

- Does the disjoint cycle technique “work”?  
- Unlikely: the following permutation has \( n/2 \) cycles of length two, but \( bid(\pi) = td(\pi) = 1 \):

\[
\begin{array}{cccc}
\frac{n}{2} + 1 & \frac{n}{2} + 2 & \frac{n}{2} + 3 & \ldots & n & 1 & 2 & 3 & \ldots & \frac{n}{2}
\end{array}
\]

- We’ll need something else since we cannot bound the effect of an operation in that setting;
The “directed breakpoint graph”

- (the term “breakpoint” will be explained later);
- Let’s build the directed breakpoint graph of \( \pi = \langle 4 \ 1 \ 6 \ 2 \ 5 \ 7 \ 3 \rangle \):

1. build the ordered vertex set \((\pi_0 = 0, \pi_1, \pi_2, \ldots, \pi_n)\);
2. add \textbf{black arcs} for every ordered pair \((\pi_i, \pi_{i-1} \ (\text{mod} \ n+1))\);
3. add \textbf{grey arcs} for every ordered pair \((i, i + 1 \ (\text{mod} \ n + 1))\);

\(\text{DBG}(\pi)\) decomposes in a unique way into \textbf{alternating cycles}
Definition ([Bafna and Pevzner, 1998])

The **directed breakpoint graph** of \( \langle \pi_1 \pi_2 \cdots \pi_n \rangle \) is defined by:

1. an ordered vertex set \( V = (\pi_0 = 0, \pi_1, \pi_2, \ldots, \pi_n) \);
2. a bicoloured arc set \( A = A_B \cup A_G \), where:
   
   2.1 \( A_B = \{(\pi_i, \pi_{i-1} \pmod{n+1}) : 0 \leq i \leq n\} \);
   
   2.2 \( A_G = \{(i, i + 1 \pmod{n+1}) : 0 \leq i \leq n\} \);
Circular vs. linear layout

- One can also represent the directed breakpoint graph using a linear layout without affecting the cycle structure:

Circular → linear: split 0 into 0 and $n + 1$;

Linear → circular: merge 0 and $n + 1$ into 0;

(in both cases: adapt arcs accordingly);
Intuitions behind $DBG(\pi) - 1$

- Both “monochromatic” cycles represent an ordering:
  1. the black one represents the one we have (“reality”);
  2. the grey one represents the one we want to obtain (“desire”);
Intuitions behind $\text{DBG}(\pi) - 2$

- The alternating cycles are a blend of “reality” and “desire”, and we must act on those cycles to turn “reality” into “desire”;
- When we are done, we have the largest number of cycles;
One way of proving lower bounds on distances: be optimistic:

1. find out the “best case”;
2. pretend we’re always in that case;

Techniques based on breakpoint graphs follow the same spirit as those based on the disjoint cycle decomposition, but less freedom is allowed (details later);

Usually: case analysis to determine by how much a parameter of the graph can change with one operation;
Lower bounding the block-interchange distance

A block-interchange $\beta(i, j, k, \ell)$ increases the number of cycles in $DBG$ by at most 2:

\[
\pi_i - 1 \pi_i \pi_j - 1 \pi_j \pi_k - 1 \pi_k \pi_\ell - 1 \pi_\ell - 1 \pi_i - 1 \pi_i \pi_j - 1 \pi_j \pi_k - 1 \pi_k \pi_\ell - 1 \pi_\ell
\]

Theorem ([Christie, 1996])

For all $\pi$ in $S_n$: $\text{bid}(\pi) \geq \frac{n+1-c(DBG(\pi))}{2}$. 
Lower bounding the transposition distance

A transposition $\tau(i,j,k)$ increases the number of odd cycles in $DBG$ by at most 2:

$$\pi_{i-1} \pi_i \pi_{j-1} \pi_j \pi_{k-1} \pi_k \rightarrow \pi_{i-1} \pi_j \pi_{k-1} \pi_j \pi_{j-1} \pi_k$$

$$\pi_{i-1} \pi_i \pi_{j-1} \pi_j \pi_{k-1} \pi_k \rightarrow \pi_{i-1} \pi_j \pi_{k-1} \pi_j \pi_{j-1} \pi_k$$

$$\pi_{i-1} \pi_i \pi_{j-1} \pi_j \pi_{k-1} \pi_k \rightarrow \pi_{i-1} \pi_j \pi_{k-1} \pi_j \pi_{j-1} \pi_k$$

Theorem ([Bafna and Pevzner, 1998])

For all $\pi$ in $S_n$: $td(\pi) \geq \frac{n+1-c_{odd}(DBG(\pi))}{2}$. 
A more general lower bounding technique [Labarre, 2013]

Recall the “monochromatic” decomposition of $DBG(\pi)$: We have $\overline{\pi} = \hat{\pi} \circ \dot{\pi}$, with $dcd(\pi) \cong G(\pi)$; indeed:

\[
\begin{align*}
\overline{\pi} = \hat{\pi} \circ \dot{\pi} &= (0, 1, 5, 3)(2, 7, 6) \\
\end{align*}
\]
Rearrangements in the setting of $\bar{\pi}$

- Bottom line: $DBG(\pi)$ is a permutation $\bar{\pi}$;
- We can translate the effect of any rearrangement $\sigma$ on $\bar{\pi}$:

**Lemma ([Labarre, 2013])**

For all $\pi, \sigma \in S_n$: $\bar{\pi} \circ \bar{\sigma} = \bar{\pi} \circ \bar{\sigma}^{\pi}$.

**Proof sketch.**

Use the definition of $\bar{\pi}$ and simplify to the wanted expression.
Conjugating permutations

Definition

Let $\pi, \sigma \in S_n$; the *conjugate* of $\pi$ by $\sigma$ is the permutation

$$\pi^\sigma = \sigma \pi \sigma^{-1}.$$  

An easy (well, less tedious) way of computing $\pi^\sigma$:

$$\pi = \begin{pmatrix} 1 & 2 & \cdots & n \\ \pi_1 & \pi_2 & \cdots & \pi_n \end{pmatrix} \quad \rightarrow \quad \pi^\sigma = \begin{pmatrix} \sigma_1 & \sigma_2 & \cdots & \sigma_n \\ \sigma_{\pi_1} & \sigma_{\pi_2} & \cdots & \sigma_{\pi_n} \end{pmatrix}$$
A lower bounding theorem

- This in turns yields:

**Theorem ([Labarre, 2013])**

Let:

1. \( S \subset S_n, \) with \( S = \{s_1, s_2, \ldots\} \),
2. \( S' = \{\overline{s_1}, \overline{s_2}, \ldots\} \),
3. \( C \) the set of conjugacy classes that intersect \( S' \).

Then for all \( \pi \) in \( S_n \), every factorisation of \( \pi \) into \( t \) elements of \( S \) yields a factorisation of \( \overline{\pi} \) into \( t \) elements of \( C \).

- And as an immediate corollary:

\[
d_S(\pi) \geq d_C(\overline{\pi})
\]
An algorithm for obtaining lower bounds

To derive a lower bound on $d_S(\pi)$, we:

1. compute $S' = \{s_1, s_2, \ldots\}$;
2. identify the set $C$ of conjugacy classes that cover $S'$;
3. use any lower bound on $d_C(\overline{\pi})$ as a lower bound on $d_S(\pi)$;
Application: lower bounding the block-interchange distance

As an example, consider the block-interchange distance:

1. if $\beta(i, j, k, \ell)$ is a block-interchange, then $\bar{\beta}(i, j, k, \ell)$ is a 3-cycle or two disjoint 2-cycles;
2. so $\mathcal{C}$ is the set of all pairs of 2-cycles;
3. since $d_\mathcal{C}(\pi) = (n + 1 - c(\pi))/2$ (Cayley’s theorem), we recover

$$d_S(\pi) \geq d_\mathcal{C}(\pi) = (n + 1 - c(\pi))/2 = \frac{n + 1 - c(DBG(\pi))}{2}$$
References I

Sorting by transpositions.

Cayley, A. (1849).
Note on the theory of permutations.

Sorting permutations by block-interchanges.

Lower bounding edit distances between permutations.